PersonalizedCancerDiagnosis

May 31, 2019

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data

- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.

- Both these data files are have a common column called ID
- Data file's information:
  training_variants (ID , Gene, Variations, Class)
  training_text (ID, Text)

2.1.2. Example Data Point
training_variants
ID,Gene,Variation,Class 0,FAM58A,Truncating Mutations,1 1,CBL,W802*,2 2,CBL,Q249E,2 . . .
training_text
ID,Text 0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). . . .

2.2. Mapping the real-world problem to an ML problem
2.2.1. Type of Machine Learning Problem

```
There are nine different classes a genetic mutation can be classified into => Multi class 
```

2.2.2. Performance Metric
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation
Metric(s): * Multi class log-loss * Confusion matrix
2.2.3. Machine Learing Objectives and Constraints
Objective: Predict the probability of each data-point belonging to each of the nine classes.
Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import re
        import time
        import warnings
        import numpy as np
        from nltk.corpus import stopwords
        from sklearn.decomposition import TruncatedSVD
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.manifold import TSNE
        import seaborn as sns
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics.classification import accuracy_score, log_loss
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.linear_model import SGDClassifier
        from imblearn.over_sampling import SMOTE
        from collections import Counter
        from scipy.sparse import hstack
        from sklearn.multiclass import OneVsRestClassifier
        from sklearn.svm import SVC
        from sklearn.model_selection import StratifiedKFold
        from collections import Counter, defaultdict
        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.naive_bayes import GaussianNB
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV
        import math
        from sklearn.metrics import normalized_mutual_info_score
        from sklearn.ensemble import RandomForestClassifier
        warnings.filterwarnings("ignore")

        from mlxtend.classifier import StackingClassifier

        from sklearn import model_selection
        from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
In [2]: data = pd.read_csv('training_variants')
        print('Number of data points : ', data.shape[0])
        print('Number of features : ', data.shape[1])
        print('Features : ', data.columns.values)
        data.head()

Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']


Out[2]:    ID    Gene             Variation  Class
        0   0  FAM58A  Truncating Mutations      1
        1   1     CBL                 W802*      2
        2   2     CBL                 Q249E      2
        3   3     CBL                 N454D      3
        4   4     CBL                 L399V      4
```

training/training_variants is a comma separated file containing the description of the genetic mutations used for training. Fields are

ID : the id of the row used to link the mutation to the clinical evidence

Gene : the gene where this genetic mutation is located

Variation : the aminoacid change for this mutations

Class : 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

```
In [3]: # note the seprator in this file
        data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],
        print('Number of data points : ', data_text.shape[0])
        print('Number of features : ', data_text.shape[1])
        print('Features : ', data_text.columns.values)
        data_text.head()

Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']


Out[3]:    ID                                               TEXT
        0   0  Cyclin-dependent kinases (CDKs) regulate a var...
        1   1   Abstract Background  Non-small cell lung canc...
        2   2   Abstract Background  Non-small cell lung canc...
        3   3  Recent evidence has demonstrated that acquired...
        4   4  Oncogenic mutations in the monomeric Casitas B...
```

3.1.3. Preprocessing of text

```
In [4]: # loading stop words from nltk library
        stop_words = set(stopwords.words('english'))


        def nlp_preprocessing(total_text, index, column):
            if type(total_text) is not int:
                string = ""
                # replace every special char with space
                total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
                # replace multiple spaces with single space
                total_text = re.sub('\s+',' ', total_text)
                # converting all the chars into lower-case.
                total_text = total_text.lower()

                for word in total_text.split():
                # if the word is a not a stop word then retain that word from the data
                    if not word in stop_words:
                        string += word + " "

                data_text[column][index] = string
```

```
In [5]: #text processing stage.
        start_time = time.clock()
        for index, row in data_text.iterrows():
            if type(row['TEXT']) is str:
                nlp_preprocessing(row['TEXT'], index, 'TEXT')
            else:
                print("there is no text description for id:",index)
        print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 127.79852770000001 seconds
```

```
In [6]: #merging both gene_variations and text data based on ID
        result = pd.merge(data, data_text,on='ID', how='left')
        result.head()
```

```
Out[6]:    ID    Gene             Variation  Class  \
        0   0  FAM58A  Truncating Mutations      1
        1   1     CBL                 W802*      2
        2   2     CBL                 Q249E      2
        3   3     CBL                 N454D      3
        4   4     CBL                 L399V      4
```

```
                                                   TEXT
0  cyclin dependent kinases cdks regulate variety...
1  abstract background non small cell lung cancer...
2  abstract background non small cell lung cancer...
3  recent evidence demonstrated acquired uniparen...
4  oncogenic mutations monomeric casitas b lineag...
```

```
In [7]: result[result.isnull().any(axis=1)]
```

```
Out[7]:         ID    Gene             Variation  Class TEXT
        1109  1109   FANCA                S1088F      1  NaN
        1277  1277  ARID5B  Truncating Mutations      1  NaN
        1407  1407   FGFR3                 K508M      6  NaN
        1639  1639    FLT1         Amplification      6  NaN
        2755  2755    BRAF                 G596C      7  NaN
```

```
In [8]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [9]: result[result['ID']==1109]
```

```
Out[9]:         ID   Gene Variation  Class          TEXT
        1109  1109  FANCA    S1088F      1  FANCA S1088F
```

3.1.4. Test, Train and Cross Validation Split
3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

```
In [10]: y_true = result['Class'].values
         result.Gene      = result.Gene.str.replace('\s+', '_')
         result.Variation = result.Variation.str.replace('\s+', '_')

         # split the data into test and train by maintaining same distribution of output varai
         X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true,
         # split the train data into train and cross validation by maintaining same distributi
         train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

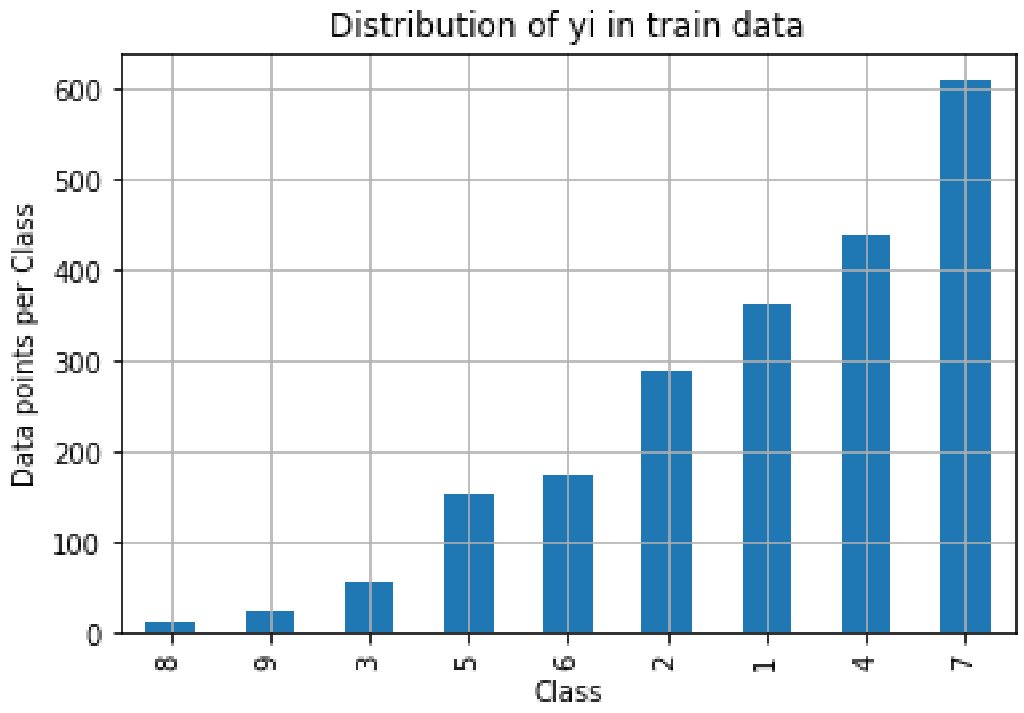
```
In [11]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

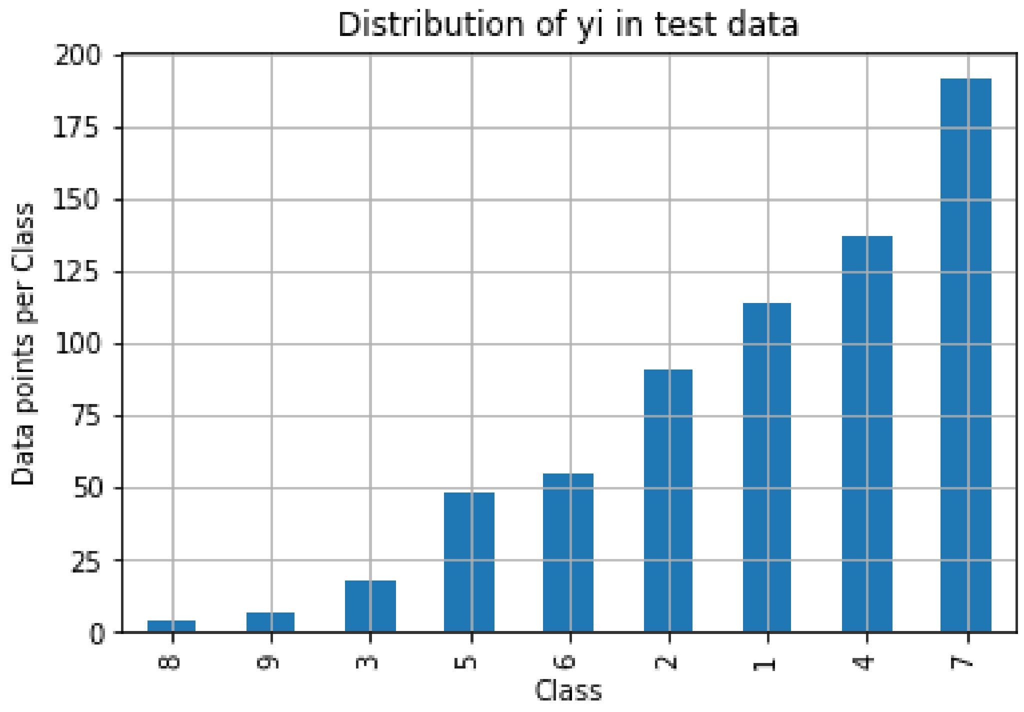
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [12]: # it returns a dict, keys as class labels and values as the number of data points in
         train_class_distribution = train_df['Class'].value_counts().sort_values()
         test_class_distribution = test_df['Class'].value_counts().sort_values()
         cv_class_distribution = cv_df['Class'].value_counts().sort_values()

         my_colors = 'rgbkymc'
         train_class_distribution.plot(kind='bar')
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in train data')
         plt.grid()
         plt.show()

         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-train_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':',train_class_distribution.values[


         print('-'*80)
         my_colors = 'rgbkymc'
         test_class_distribution.plot(kind='bar')
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in test data')
         plt.grid()
         plt.show()

         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-test_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':',test_class_distribution.values[i

         print('-'*80)
         my_colors = 'rgbkymc'
         cv_class_distribution.plot(kind='bar')
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in cross validation data')
         plt.grid()
         plt.show()

         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-train_class_distribution.values)
         for i in sorted_yi:
```

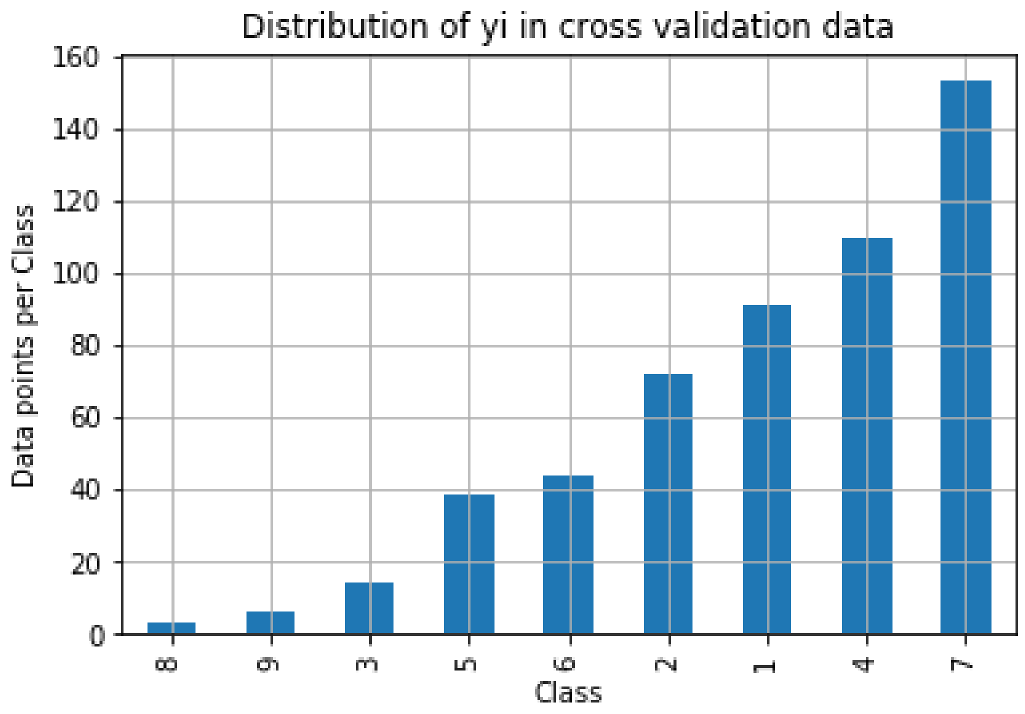
```
        print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],
```



```
Number of data points in class 9 : 609 ( 28.672 %)
Number of data points in class 8 : 439 ( 20.669 %)
Number of data points in class 7 : 363 ( 17.09 %)
Number of data points in class 6 : 289 ( 13.606 %)
Number of data points in class 5 : 176 ( 8.286 %)
Number of data points in class 4 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 2 : 24 ( 1.13 %)
Number of data points in class 1 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class 9 : 191 ( 28.722 %)
Number of data points in class 8 : 137 ( 20.602 %)
Number of data points in class 7 : 114 ( 17.143 %)
Number of data points in class 6 : 91 ( 13.684 %)
Number of data points in class 5 : 55 ( 8.271 %)
Number of data points in class 4 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 2 : 7 ( 1.053 %)
Number of data points in class 1 : 4 ( 0.602 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class 9 : 153 ( 28.759 %)
Number of data points in class 8 : 110 ( 20.677 %)
Number of data points in class 7 : 91 ( 17.105 %)
Number of data points in class 6 : 72 ( 13.534 %)
Number of data points in class 5 : 44 ( 8.271 %)
Number of data points in class 4 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 2 : 6 ( 1.128 %)
Number of data points in class 1 : 3 ( 0.564 %)
```

3.2 Prediction using a ‘Random’ Model

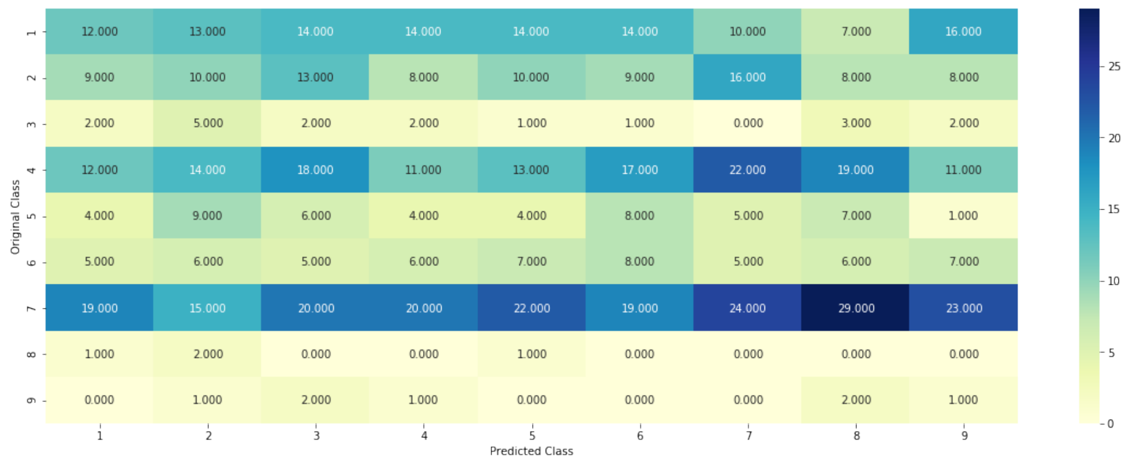
In a ‘Random’ Model, we generate the NINE class probabilites randomly such that they sum to 1.

```
In [13]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
             # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

             A =(((C.T)/(C.sum(axis=1))).T)
             #divid each element of the confusion matrix with the sum of elements in that colu

             # C = [[1, 2],
             #     [3, 4]]
             # C.T = [[1, 3],
             #        [2, 4]]
             # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in
             # C.sum(axix =1) = [[3, 7]]
             # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
             #                            [2/3, 4/7]]

             # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
             #                            [3/7, 4/7]]
             # sum of row elements = 1

             B =(C/C.sum(axis=0))
             #divid each element of the confusion matrix with the sum of elements in that row
             # C = [[1, 2],
             #     [3, 4]]
             # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in
             # C.sum(axix =0) = [[4, 6]]
             # (C/C.sum(axis=0)) = [[1/4, 2/6],
             #                      [3/4, 4/6]]

             labels = [1,2,3,4,5,6,7,8,9]
             # representing A in heatmap format
             print("-"*20, "Confusion matrix", "-"*20)
             plt.figure(figsize=(20,7))
             sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabe
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.show()

             print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
             plt.figure(figsize=(20,7))
             sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabe
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.show()

             # representing B in heatmap format
             print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
             plt.figure(figsize=(20,7))
             sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabe
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.show()
```

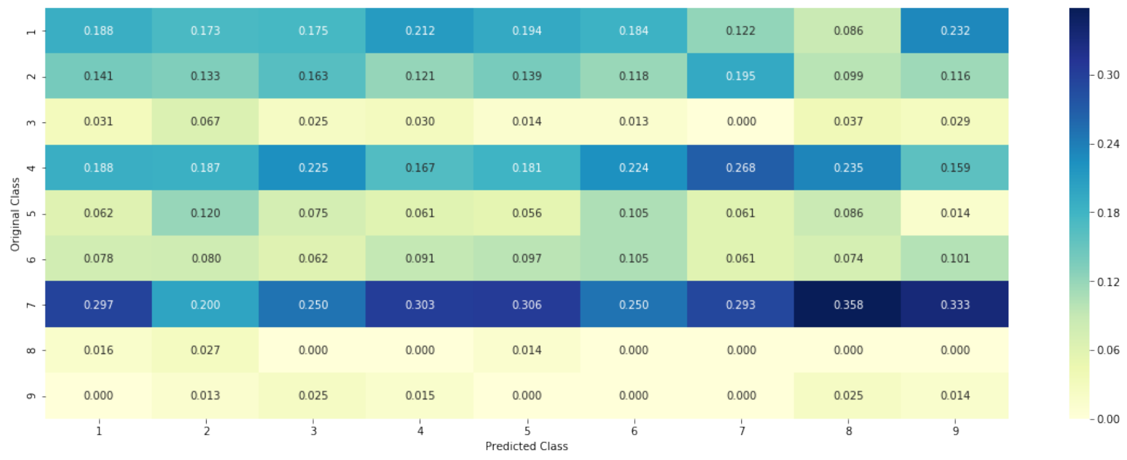
```
In [14]: # we need to generate 9 numbers and the sum of numbers should be 1
         # one solution is to genarate 9 numbers and divide each of the numbers by their sum
         # ref: https://stackoverflow.com/a/18662466/4084039
```

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predict


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

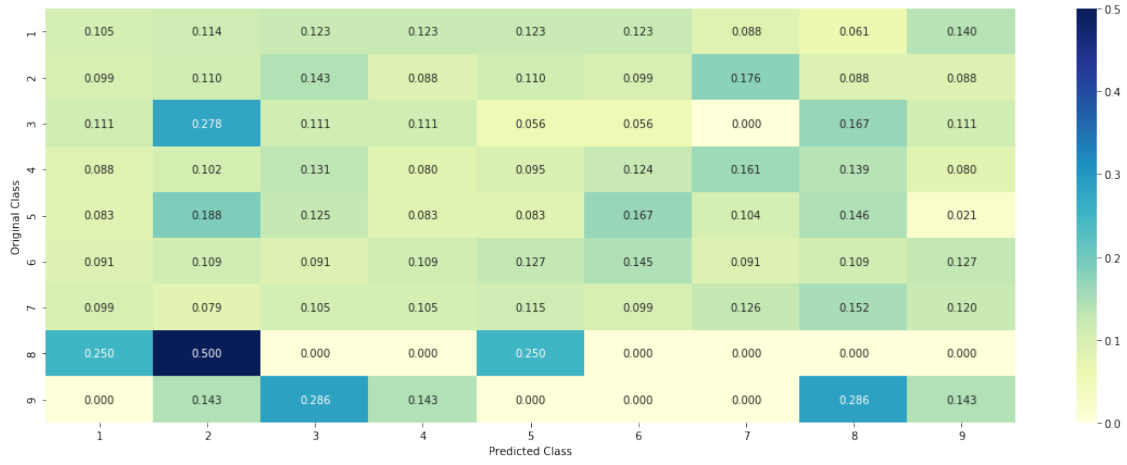
```
Log loss on Cross Validation Data using Random Model 2.4502872848706745
Log loss on Test Data using Random Model 2.5324097049245133
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



3.3 Univariate Analysis

```
In [15]: # code for response coding with Laplace smoothing.
         # alpha : used for laplace smoothing
         # feature: ['gene', 'variation']
         # df: ['train_df', 'test_df', 'cv_df']
         # algorithm
         # ----------
         # Consider all unique values and the number of occurances of given feature in train d
         # build a vector (1*9) , the first element = (number of times it occured in class1 + 
         # gv_dict is like a look up table, for every gene it store a (1*9) representation of
         # for a value of feature in df:
         # if it is in train data:
```

```
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #         {BRCA1      174
    #          TP53       106
    #          EGFR        86
    #          BRCA2       75
    #          PTEN        69
    #          KIT         61
    #          BRAF        60
    #          ERBB2       47
    #          PDGFRA      46
    #          ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each ge
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in t
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pert
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')]
            #         ID   Gene             Variation  Class
```

```
                # 2470  2470  BRCA1               S1715C      1
                # 2486  2486  BRCA1               S1841R      1
                # 2614  2614  BRCA1                  M1R      1
                # 2432  2432  BRCA1               L1657P      1
                # 2567  2567  BRCA1               T1685A      1
                # 2583  2583  BRCA1               E1660G      1
                # 2634  2634  BRCA1               W1718L      1
                # cls_cnt.shape[0] will return the number of rows

                cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

                # cls_cnt.shape[0](numerator) will contain the number of time that partic
                vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

            # we are adding the gene/variation to the dict as key and vec as value
            gv_dict[i]=vec
        return gv_dict

    # Get Gene variation feature
    def get_gv_feature(alpha, feature, df):
        # print(gv_dict)
        #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 
        #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 
        #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181
        #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
        #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
        #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0
        #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 
        #      ...
        #     }
        gv_dict = get_gv_fea_dict(alpha, feature, df)
        # value_count is similar in get_gv_fea_dict
        value_count = train_df[feature].value_counts()

        # gv_fea: Gene_variation feature, it will contain the feature for each feature va
        gv_fea = []
        # for every feature values in the given data frame we will check if it is there i
        # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
        for index, row in df.iterrows():
            if row[feature] in dict(value_count).keys():
                gv_fea.append(gv_dict[row[feature]])
            else:
                gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#                 gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
        return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

(numerator + 10*alpha) / (denominator + 90*alpha)
3.2.1 Univariate Analysis on Gene Feature
Q1. Gene, What type of feature it is ?
Ans. Gene is a categorical variable
Q2. How many categories are there and How they are distributed?

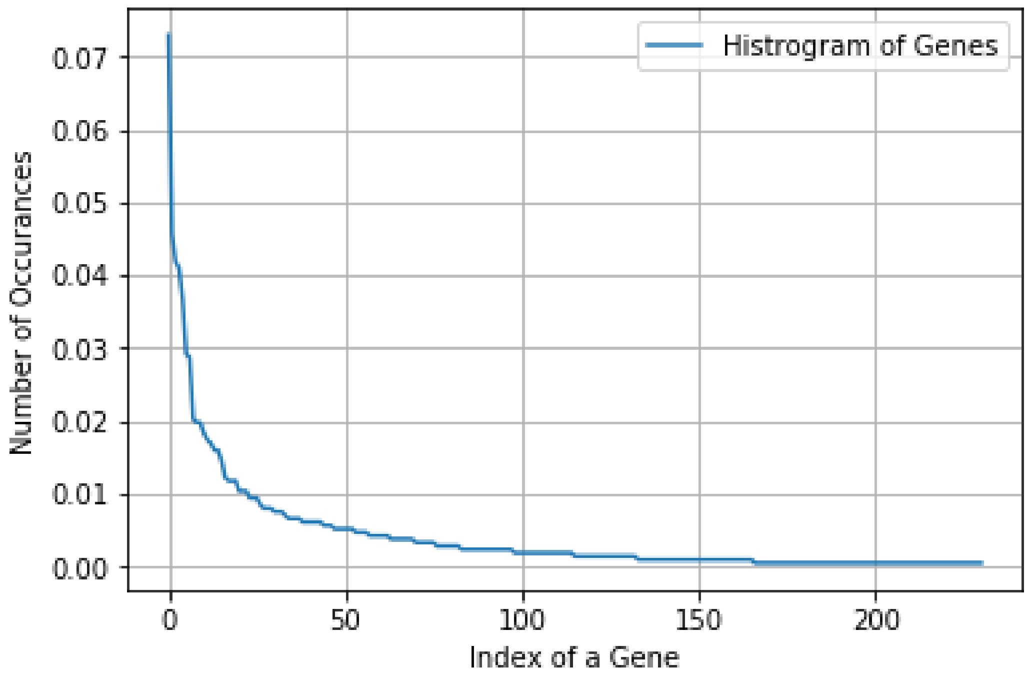
```
In [16]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occured most
         print(unique_genes.head(10))
```

```
Number of Unique Genes : 231
BRCA1     155
TP53       97
BRCA2      89
EGFR       87
PTEN       78
BRAF       62
KIT        61
ALK        43
ERBB2      42
PDGFRA     42
Name: Gene, dtype: int64
```

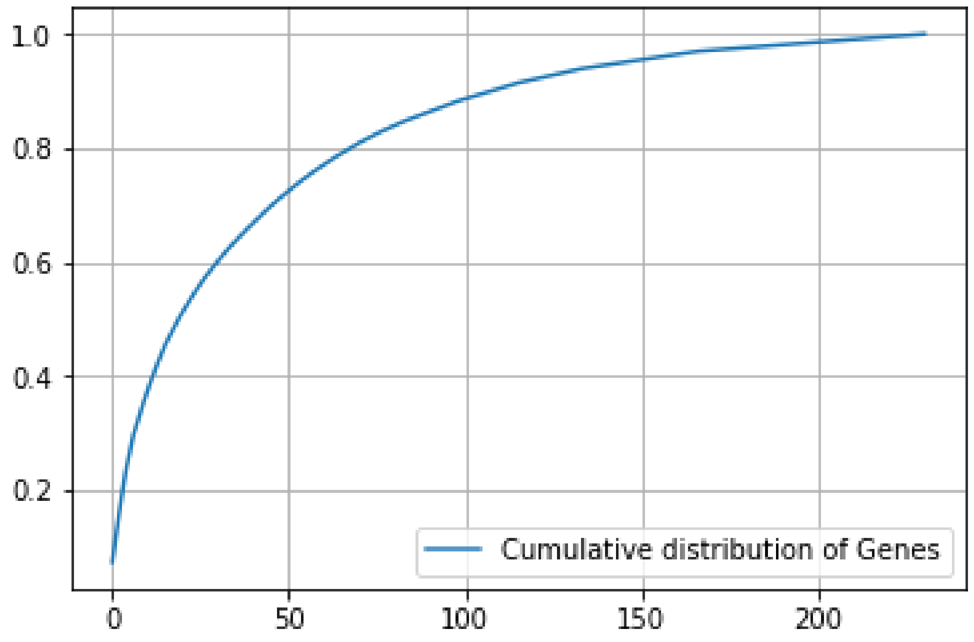
```
In [17]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the
```

```
Ans: There are 231 different categories of genes in the train data, and they are distibuted as
```

```
In [18]: s = sum(unique_genes.values);
         h = unique_genes.values/s;
         plt.plot(h, label="Histrogram of Genes")
         plt.xlabel('Index of a Gene')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.grid()
         plt.show()
```



```
In [19]: c = np.cumsum(h)
         plt.plot(c,label='Cumulative distribution of Genes')
         plt.grid()
         plt.legend()
         plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

```
In [20]: #response-coding of the Gene feature
         # alpha is used for laplace smoothing
         alpha = 1
         # train gene feature
         train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
         # test gene feature
         test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
         # cross validation gene feature
         cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))

In [21]: print("train_gene_feature_responseCoding is converted feature using respone coding me

train_gene_feature_responseCoding is converted feature using respone coding method. The shape 


In [22]: # one-hot encoding of Gene feature.
         gene_vectorizer = CountVectorizer()
         train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
         test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
         cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

In [23]: train_df['Gene'].head()

Out[23]: 104      MSH6
         3291      RET
         2303     JAK1
         2398      NF1
         1063    EWSR1
         Name: Gene, dtype: object

In [24]: gene_vectorizer.get_feature_names()

Out[24]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
'akt3',
'alk',
'apc',
'ar',
'araf',
'arid1b',
'arid5b',
'asxl1',
'atm',
'atr',
'atrx',
'aurka',
'axin1',
'b2m',
'bap1',
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gli1',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'jak1',
'jak2',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhc',
'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

```
In [25]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding me
```

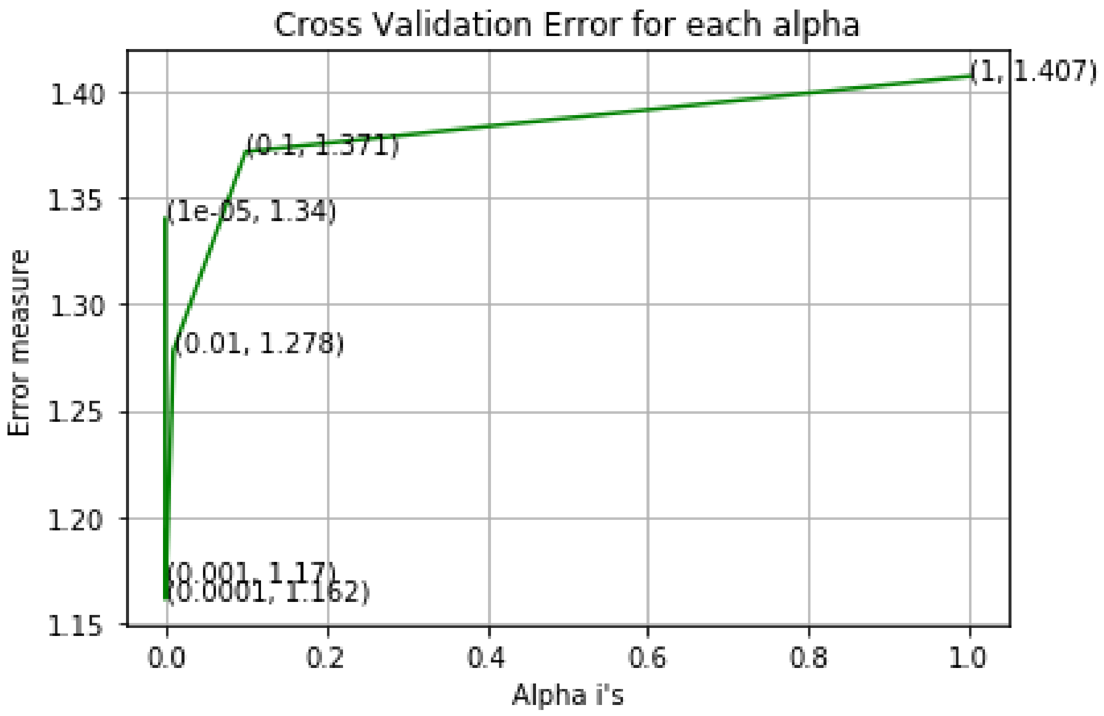
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape 
```

Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

```
In [50]: alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link:
         #------------------------------


         cv_log_error_array=[]
         for i in alpha:
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(train_gene_feature_onehotCoding, y_train)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_gene_feature_onehotCoding, y_train)
             predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
             cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
             print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

         fig, ax = plt.subplots()
         ax.plot(alpha, cv_log_error_array,c='g')
         for i, txt in enumerate(np.round(cv_log_error_array,3)):
             ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
         plt.grid()
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()


         best_alpha = np.argmin(cv_log_error_array)
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         clf.fit(train_gene_feature_onehotCoding, y_train)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_gene_feature_onehotCoding, y_train)

         predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
         predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
         predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.3399240641735715
For values of alpha =  0.0001 The log loss is: 1.1615127881835052
For values of alpha =  0.001 The log loss is: 1.1699407402866273
For values of alpha =  0.01 The log loss is: 1.2780645980559562
For values of alpha =  0.1 The log loss is: 1.3714411995726248
For values of alpha =  1 The log loss is: 1.406814257486051
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0468088389799284
For values of best alpha =  0.0001 The cross validation log loss is: 1.1615127881835052
For values of best alpha =  0.0001 The test log loss is: 1.2506024934888313
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

```
In [51]: print("Q6. How many data points in Test and CV datasets are covered by the ", unique_

         test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
         cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_cove
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_co
```

```
Q6. How many data points in Test and CV datasets are covered by the  231  genes in train datas
Ans
```

```
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 508 out of  532 : 95.48872180451127
```

3.2.2 Univariate Analysis on Variation Feature
Q7. Variation, What type of feature is it ?
Ans. Variation is a categorical variable
Q8. How many categories are there?

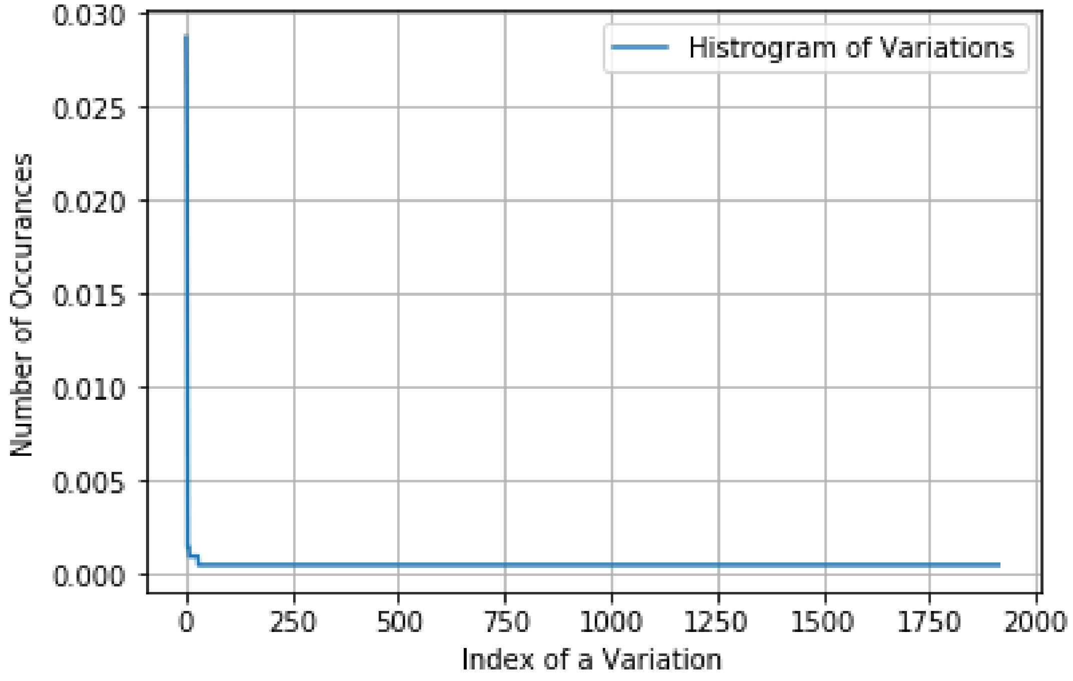
```
In [52]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1914
Truncating_Mutations    61
Deletion                54
Amplification           49
Fusions                 21
G12V                     3
T58I                     3
E17K                     3
Overexpression           3
A146T                    2
Q61R                     2
Name: Variation, dtype: int64
```

```
In [53]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

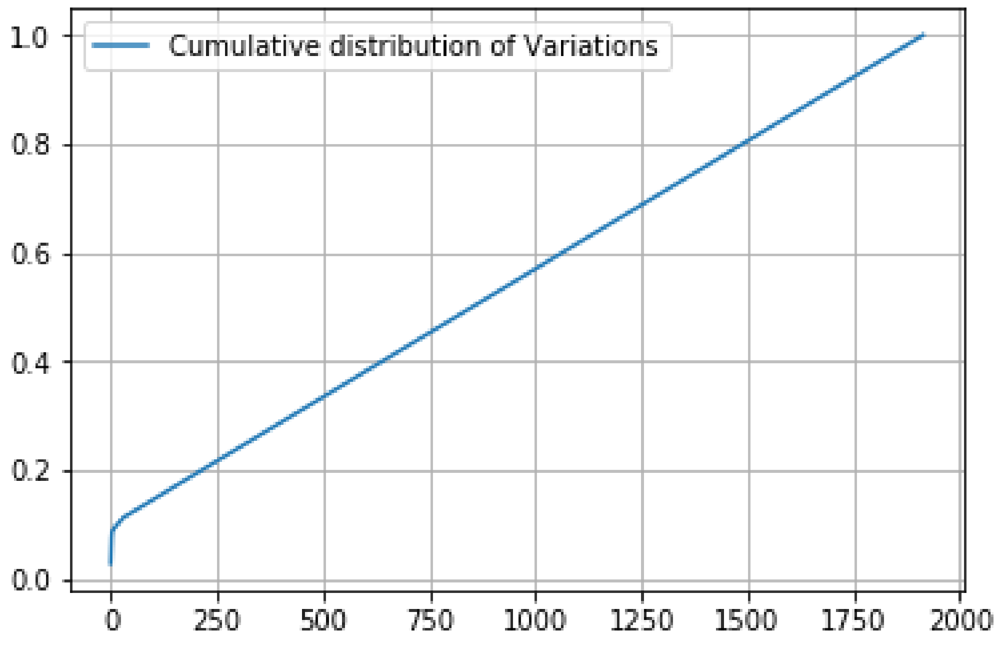
```
Ans: There are 1914 different categories of variations in the train data, and they are distibu
```

```
In [54]: s = sum(unique_variations.values);
         h = unique_variations.values/s;
         plt.plot(h, label="Histrogram of Variations")
         plt.xlabel('Index of a Variation')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.grid()
         plt.show()
```



```
In [55]: c = np.cumsum(h)
         print(c)
         plt.plot(c,label='Cumulative distribution of Variations')
         plt.grid()
         plt.legend()
         plt.show()
```

```
[0.0287194  0.05414313 0.07721281 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans.There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

Response coding

We will be using both these methods to featurize the Variation Feature

```
In [56]: # alpha is used for laplace smoothing
         alpha = 1
         # train gene feature
         train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
         # test gene feature
         test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
         # cross validation gene feature
         cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_d
```

```
In [57]: print("train_variation_feature_responseCoding is a converted feature using the respons
```

```
train_variation_feature_responseCoding is a converted feature using the response coding method
```

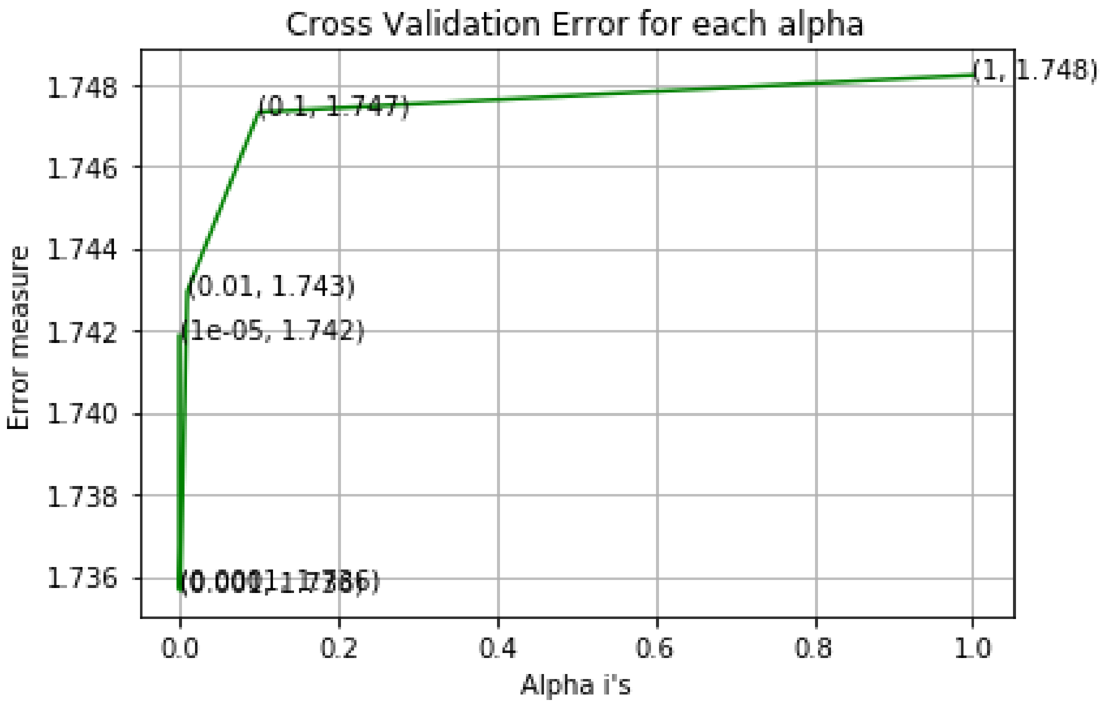
```
In [58]: # one-hot encoding of variation feature.
         variation_vectorizer = CountVectorizer()
         train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
         test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
         cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation']]
In [59]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e

train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

```
In [60]: alpha = [10 ** x for x in range(-5, 1)]

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link:
         #------------------------------


         cv_log_error_array=[]
         for i in alpha:
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(train_variation_feature_onehotCoding, y_train)

             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_variation_feature_onehotCoding, y_train)
             predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

             cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
             print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

         fig, ax = plt.subplots()
         ax.plot(alpha, cv_log_error_array,c='g')
         for i, txt in enumerate(np.round(cv_log_error_array,3)):
             ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
         plt.grid()
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.7418651459567152
For values of alpha =  0.0001 The log loss is: 1.7357542586616825
For values of alpha =  0.001 The log loss is: 1.7356621854637921
For values of alpha =  0.01 The log loss is: 1.742935095734731
For values of alpha =  0.1 The log loss is: 1.747315893637275
For values of alpha =  1 The log loss is: 1.7482170935886212
```



```
For values of best alpha =  0.001 The train log loss is: 1.082397937942143
For values of best alpha =  0.001 The cross validation log loss is: 1.7356621854637921
For values of best alpha =  0.001 The test log loss is: 1.7083793647338343
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Not sure! But lets be very sure using the below analysis.

```
In [61]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], '
         test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
         cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_cove
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_co
```

```
Q12. How many data points are covered by total  1914  genes in test and cross validation data s
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 46 out of  532 : 8.646616541353383
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

```
In [62]: # cls_text is a data frame
         # for every row in data fram consider the 'TEXT'
         # split the words by space
         # make a dict with those words
         # increment its count whenever we see that word

         def extract_dictionary_paddle(cls_text):
             dictionary = defaultdict(int)
             for index, row in cls_text.iterrows():
                 for word in row['TEXT'].split():
                     dictionary[word] +=1
             return dictionary
```

```
In [63]: import math
         #https://stackoverflow.com/a/1602964
         def get_text_responsecoding(df):
             text_feature_responseCoding = np.zeros((df.shape[0],9))
             for i in range(0,9):
                 row_index = 0
                 for index, row in df.iterrows():
                     sum_prob = 0
                     for word in row['TEXT'].split():
                         sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(
                     text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TE
                     row_index += 1
             return text_feature_responseCoding
```

```
In [64]: # building a CountVectorizer with all the words that occured minimum 3 times in train
         text_vectorizer = TfidfVectorizer(ngram_range=(1,4), max_features=1000)
         train_text_feature_tfidf = text_vectorizer.fit_transform(train_df['TEXT'])
         # getting all the feature names (words)
         train_text_features= text_vectorizer.get_feature_names()

         # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*num
         train_text_fea_counts = train_text_feature_tfidf.sum(axis=0).A1

         # zip(list(text_features),text_fea_counts) will zip a word with its number of times i
         text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


         print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 1000
```

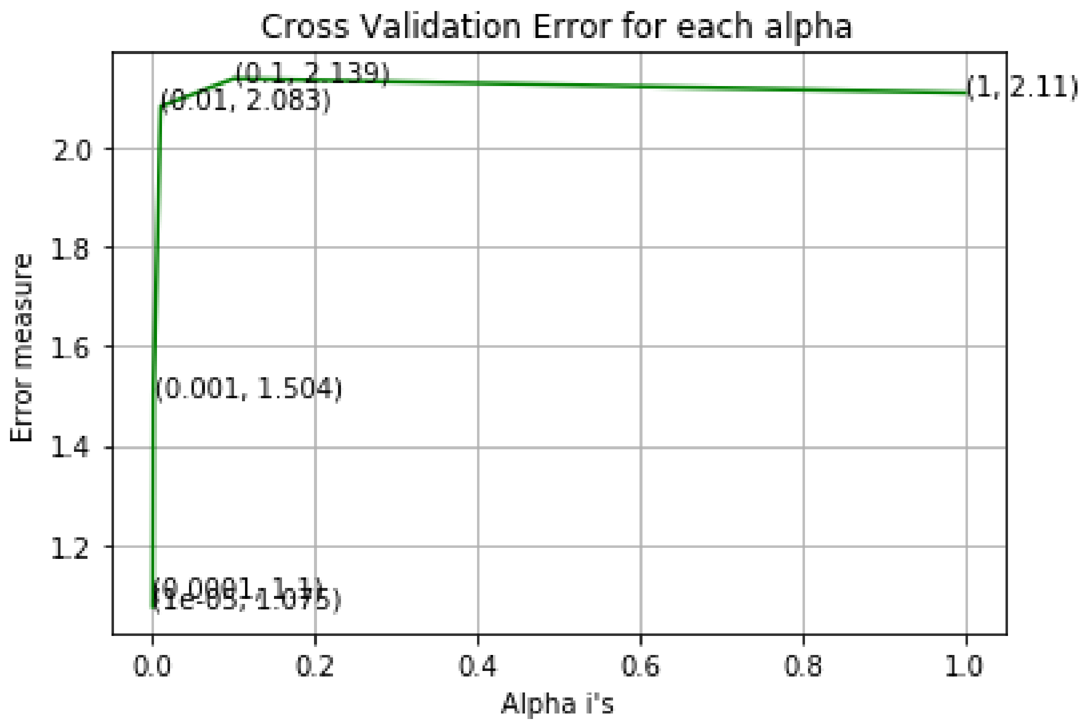
```
In [65]: dict_list = []
         # dict_list =[] contains 9 dictionaries each corresponds to a class
         for i in range(1,10):
             cls_text = train_df[train_df['Class']==i]
             # build a word dict based on the words in that class
             dict_list.append(extract_dictionary_paddle(cls_text))
             # append it to dict_list

         # dict_list[i] is build on i'th  class text data
         # total_dict is buid on whole training text data
         total_dict = extract_dictionary_paddle(train_df)


         confuse_array = []
         for i in train_text_features:
             ratios = []
             max_val = -1
             for j in range(0,9):
                 ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
             confuse_array.append(ratios)
         confuse_array = np.array(confuse_array)
```

```
In [66]: #response coding of text features
         train_text_feature_responseCoding  = get_text_responsecoding(train_df)
         test_text_feature_responseCoding  = get_text_responsecoding(test_df)
         cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

```
In [67]: # https://stackoverflow.com/a/16202486
         # we convert each row values such that they sum to 1
         train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_fe
         test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
         cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res

In [68]: # don't forget to normalize every feature
         train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)

         # we use the same vectorizer that was trained on train data
         test_text_feature_tfidf = text_vectorizer.transform(test_df['TEXT'])
         # don't forget to normalize every feature
         test_text_feature_tfidf = normalize(test_text_feature_tfidf, axis=0)

         # we use the same vectorizer that was trained on train data
         cv_text_feature_tfidf = text_vectorizer.transform(cv_df['TEXT'])
         # don't forget to normalize every feature
         cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)

In [69]: #https://stackoverflow.com/a/2258273/4084039
         sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
         sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))

In [70]: # Number of words for a given frequency.
         print(Counter(sorted_text_occur))

Counter({12.737787960916615: 2, 12.300583521924045: 2, 11.19745323817686: 2, 9.680777009569839


In [71]: # Train a Logistic regression+Calibration model using text features whicha re on-hot
         alpha = [10 ** x for x in range(-5, 1)]

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link:
         #-------------------------------


         cv_log_error_array=[]
```

```
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_tfidf, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
clf.fit(train_text_feature_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_tfidf, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.0753076052805342
For values of alpha =  0.0001 The log loss is: 1.0995377909294202
For values of alpha =  0.001 The log loss is: 1.5041404262111484
For values of alpha =  0.01 The log loss is: 2.082842339759065
For values of alpha =  0.1 The log loss is: 2.1387096243186074
For values of alpha =  1 The log loss is: 2.110319965891209
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7715694769859365
For values of best alpha =  1e-05 The cross validation log loss is: 1.0753076052805342
For values of best alpha =  1e-05 The test log loss is: 1.2069376357777855
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

```
In [72]: def get_intersec_text(df):
             df_text_vec = TfidfVectorizer(ngram_range=(1,4), max_features=1000)
             df_text_fea = df_text_vec.fit_transform(df['TEXT'])
             df_text_features = df_text_vec.get_feature_names()

             df_text_fea_counts = df_text_fea.sum(axis=0).A1
             df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
             len1 = len(set(df_text_features))
             len2 = len(set(train_text_features) & set(df_text_features))
             return len1,len2
```

```
In [73]: len1,len2 = get_intersec_text(test_df)
         print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
         len1,len2 = get_intersec_text(cv_df)
         print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train
95.0 % of word of test data appeared in train data
94.3 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
In [74]: #Data preparation for ML models.

         #Misc. functionns for ML models


         def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
             clf.fit(train_x, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x, train_y)
             pred_y = sig_clf.predict(test_x)

             # for calculating log_loss we willl provide the array of probabilities belongs to
             print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
             # calculating the number of data points that are misclassified
             print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/tes
             plot_confusion_matrix(test_y, pred_y)

In [75]: def report_log_loss(train_x, train_y, test_x, test_y,  clf):
             clf.fit(train_x, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x, train_y)
             sig_clf_probs = sig_clf.predict_proba(test_x)
             return log_loss(test_y, sig_clf_probs, eps=1e-15)

In [76]: # this function will be used just for naive bayes
         # for the given indices, we will print the name of the features
         # and we will check whether the feature present in the test point text or not
         def get_impfeature_names(indices, text, gene, var, no_features):
             gene_count_vec = CountVectorizer()
             var_count_vec = CountVectorizer()
             text_count_vec = TfidfVectorizer(ngram_range=(1,4), max_features=1000)

             gene_vec = gene_count_vec.fit(train_df['Gene'])
             var_vec  = var_count_vec.fit(train_df['Variation'])
             text_vec = text_count_vec.fit(train_df['TEXT'])

             fea1_len = len(gene_vec.get_feature_names())
             fea2_len = len(var_count_vec.get_feature_names())

             word_present = 0
             for i,v in enumerate(indices):
                 if (v < fea1_len):
                     word = gene_vec.get_feature_names()[v]
```

```
                yes_no = True if word == gene else False
                if yes_no:
                    word_present += 1
                    print(i, "Gene feature [{}] present in test data point [{}]".format(wo
            elif (v < fea1_len+fea2_len):
                word = var_vec.get_feature_names()[v-(fea1_len)]
                yes_no = True if word == var else False
                if yes_no:
                    word_present += 1
                    print(i, "variation feature [{}] present in test data point [{}]".forn
            else:
                word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
                yes_no = True if word in text.split() else False
                if yes_no:
                    word_present += 1
                    print(i, "Text feature [{}] present in test data point [{}]".format(wo

        print("Out of the top ",no_features," features ", word_present, "are present in qu
```

Stacking the three types of features

```
In [77]: # merging gene, variance and text features

         # building train, test and cross validation data sets
         # a = [[1, 2],
         #      [3, 4]]
         # b = [[4, 5],
         #      [6, 7]]
         # hstack(a, b) = [[1, 2, 4, 5],
         #                 [ 3, 4, 6, 7]]

         train_gene_var_tfidf = hstack((train_gene_feature_onehotCoding,train_variation_feature
         test_gene_var_tfidf = hstack((test_gene_feature_onehotCoding,test_variation_feature_on
         cv_gene_var_tfidf = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCo

         train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf)).tocsr()
         train_y = np.array(list(train_df['Class']))

         test_x_tfidf = hstack((test_gene_var_tfidf, test_text_feature_tfidf)).tocsr()
         test_y = np.array(list(test_df['Class']))

         cv_x_tfidf = hstack((cv_gene_var_tfidf, cv_text_feature_tfidf)).tocsr()
         cv_y = np.array(list(cv_df['Class']))


         train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
         test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variat
         cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fe
         train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_
         test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_res
         cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response
```

```
In [78]: print("One hot encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_tfidf.s
         print("(number of data points * number of features) in test data = ", test_x_tfidf.sha
         print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3171)
(number of data points * number of features) in test data =  (665, 3171)
(number of data points * number of features) in cross validation data = (532, 3171)
```

```
In [79]: print(" Response encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_respons
         print("(number of data points * number of features) in test data = ", test_x_response
         print("(number of data points * number of features) in cross validation data =", cv_x_
```

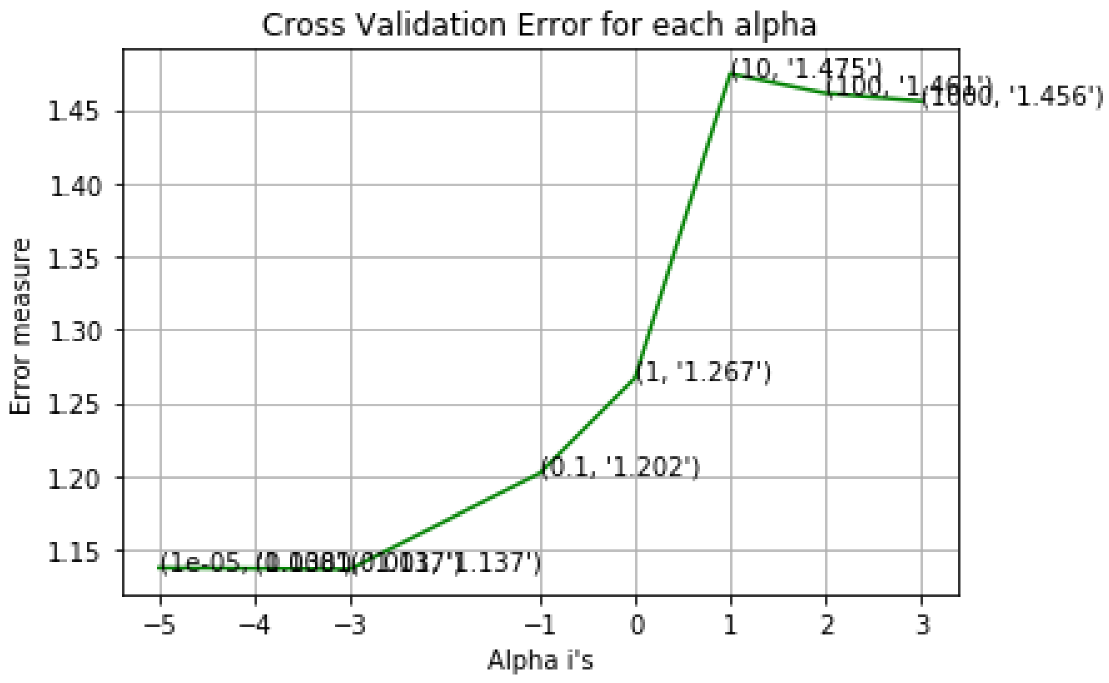
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [80]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable
         # -------------------------
         # default paramters
         # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

         # some of methods of MultinomialNB()
         # fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X, y
         # predict(X)        Perform classification on an array of test vectors X.
         # predict_log_proba(X)        Return log-probability estimates for the test vector X.
         # -----------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
         # -----------------------


         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ------------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         # ----------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
         # -----------------------


         alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = MultinomialNB(alpha=i)
             clf.fit(train_x_tfidf, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidf, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

         fig, ax = plt.subplots()
         ax.plot(np.log10(alpha), cv_log_error_array,c='g')
         for i, txt in enumerate(np.round(cv_log_error_array,3)):
             ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
         plt.grid()
         plt.xticks(np.log10(alpha))
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()


         best_alpha = np.argmin(cv_log_error_array)
         clf = MultinomialNB(alpha=alpha[best_alpha])
         clf.fit(train_x_tfidf, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_tfidf, train_y)


         predict_y = sig_clf.predict_proba(train_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
         predict_y = sig_clf.predict_proba(cv_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
         predict_y = sig_clf.predict_proba(test_x_tfidf)
```

```
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-05
Log Loss : 1.1376847263865777
for alpha = 0.0001
Log Loss : 1.1372782949880809
for alpha = 0.001
Log Loss : 1.1367449679759856
for alpha = 0.1
Log Loss : 1.202066826047302
for alpha = 1
Log Loss : 1.2670405292421814
for alpha = 10
Log Loss : 1.474555994938525
for alpha = 100
Log Loss : 1.4612505198545598
for alpha = 1000
Log Loss : 1.4558485799085321
```

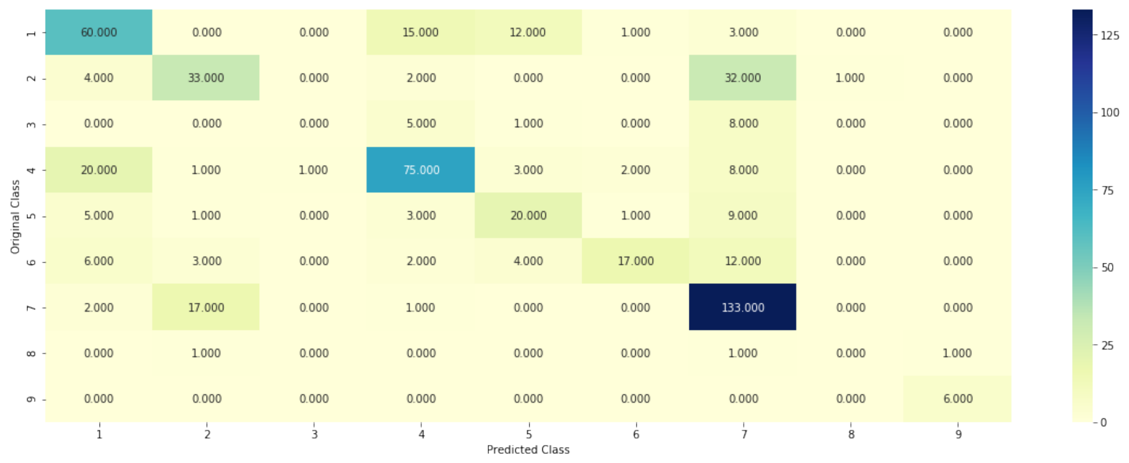


```
For values of best alpha =  0.001 The train log loss is: 0.5382422977172097
For values of best alpha =  0.001 The cross validation log loss is: 1.1367449679759856
For values of best alpha =  0.001 The test log loss is: 1.2267938061826136
```

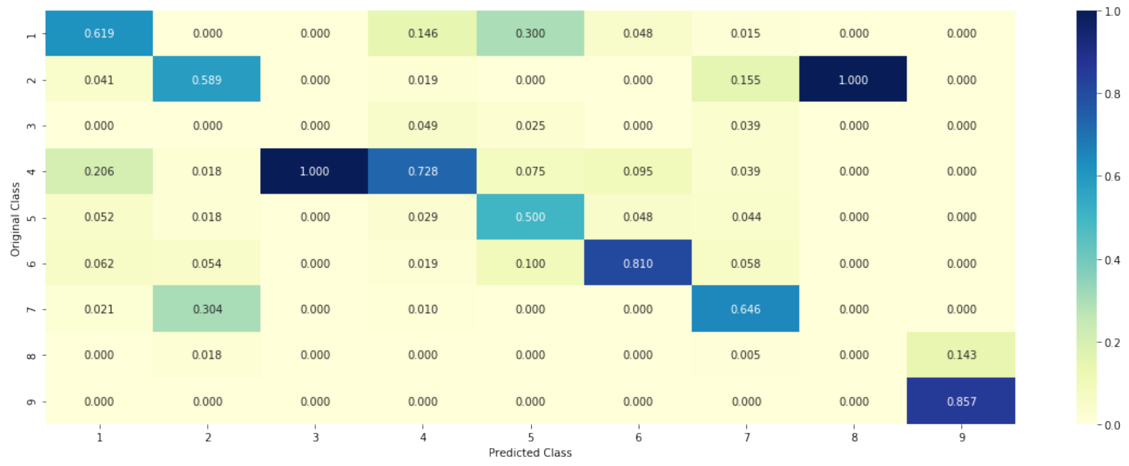
4.1.1.2. Testing the model with best hyper paramters

```
In [81]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/
         # -------------------------
         # default paramters
         # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

         # some of methods of MultinomialNB()
         # fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X, y
         # predict(X)        Perform classification on an array of test vectors X.
         # predict_log_proba(X)        Return log-probability estimates for the test vector X.
         # -----------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
         # -----------------------


         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         # ----------------------------

         clf = MultinomialNB(alpha=alpha[best_alpha])
         clf.fit(train_x_tfidf, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_tfidf, train_y)
         sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
         # to avoid rounding error while multiplying probabilites we use log-probability estim
         print("Log Loss :",log_loss(cv_y, sig_clf_probs))
         print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfid
         plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf.toarray()))
```

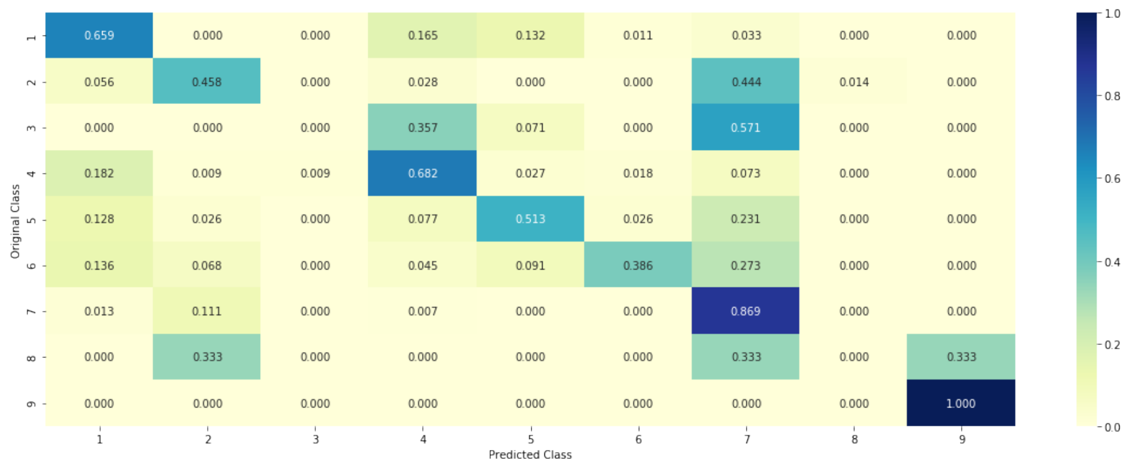
```
Log Loss : 1.1367449679759856
Number of missclassified point : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.1.1.3. Feature Importance, Correctly classified point

```
In [82]: test_point_index = 1
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3662 0.2855 0.0124 0.0787 0.0366 0.033  0.1807 0.0037 0.003
Actual Class : 1
--------------------------------------------------
7 Text feature [one] present in test data point [True]
10 Text feature [results] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [two] present in test data point [True]
14 Text feature [type] present in test data point [True]
15 Text feature [also] present in test data point [True]
18 Text feature [table] present in test data point [True]
19 Text feature [loss] present in test data point [True]
20 Text feature [region] present in test data point [True]
22 Text feature [specific] present in test data point [True]
24 Text feature [wild] present in test data point [True]
25 Text feature [however] present in test data point [True]
26 Text feature [role] present in test data point [True]
34 Text feature [may] present in test data point [True]
36 Text feature [gene] present in test data point [True]
37 Text feature [result] present in test data point [True]
```

```
38 Text feature [using] present in test data point [True]
39 Text feature [analysis] present in test data point [True]
40 Text feature [binding] present in test data point [True]
41 Text feature [suggest] present in test data point [True]
42 Text feature [affect] present in test data point [True]
43 Text feature [dna] present in test data point [True]
44 Text feature [shown] present in test data point [True]
45 Text feature [either] present in test data point [True]
50 Text feature [three] present in test data point [True]
51 Text feature [proteins] present in test data point [True]
53 Text feature [large] present in test data point [True]
54 Text feature [compared] present in test data point [True]
55 Text feature [present] present in test data point [True]
62 Text feature [well] present in test data point [True]
63 Text feature [deletion] present in test data point [True]
65 Text feature [15] present in test data point [True]
66 Text feature [similar] present in test data point [True]
67 Text feature [performed] present in test data point [True]
68 Text feature [addition] present in test data point [True]
70 Text feature [different] present in test data point [True]
71 Text feature [total] present in test data point [True]
73 Text feature [although] present in test data point [True]
74 Text feature [including] present in test data point [True]
75 Text feature [used] present in test data point [True]
76 Text feature [fig] present in test data point [True]
77 Text feature [previous] present in test data point [True]
78 Text feature [significantly] present in test data point [True]
81 Text feature [observed] present in test data point [True]
82 Text feature [expression] present in test data point [True]
84 Text feature [identified] present in test data point [True]
85 Text feature [data] present in test data point [True]
87 Text feature [mutation] present in test data point [True]
89 Text feature [additional] present in test data point [True]
90 Text feature [several] present in test data point [True]
91 Text feature [mutations] present in test data point [True]
92 Text feature [directly] present in test data point [True]
94 Text feature [thus] present in test data point [True]
95 Text feature [10] present in test data point [True]
99 Text feature [studies] present in test data point [True]
Out of the top  100  features  55 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [83]: test_point_index = 100
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
```

```
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0781 0.0547 0.013  0.0844 0.6093 0.0361 0.1172 0.0038 0.0035
Actual Class : 5
--------------------------------------------------
2 Text feature [assays] present in test data point [True]
3 Text feature [functional] present in test data point [True]
7 Text feature [neutral] present in test data point [True]
8 Text feature [variants] present in test data point [True]
9 Text feature [assay] present in test data point [True]
11 Text feature [brca1] present in test data point [True]
12 Text feature [intermediate] present in test data point [True]
17 Text feature [likely] present in test data point [True]
18 Text feature [research] present in test data point [True]
19 Text feature [based] present in test data point [True]
20 Text feature [large] present in test data point [True]
21 Text feature [variant] present in test data point [True]
22 Text feature [assess] present in test data point [True]
23 Text feature [introduction] present in test data point [True]
24 Text feature [effect] present in test data point [True]
25 Text feature [author] present in test data point [True]
26 Text feature [results] present in test data point [True]
27 Text feature [tested] present in test data point [True]
28 Text feature [assessment] present in test data point [True]
30 Text feature [used] present in test data point [True]
31 Text feature [published] present in test data point [True]
32 Text feature [available] present in test data point [True]
34 Text feature [manuscript] present in test data point [True]
35 Text feature [data] present in test data point [True]
36 Text feature [pathogenic] present in test data point [True]
37 Text feature [brct] present in test data point [True]
38 Text feature [known] present in test data point [True]
39 Text feature [controls] present in test data point [True]
40 Text feature [methods] present in test data point [True]
41 Text feature [information] present in test data point [True]
42 Text feature [sensitivity] present in test data point [True]
43 Text feature [sequence] present in test data point [True]
44 Text feature [vitro] present in test data point [True]
45 Text feature [correlation] present in test data point [True]
46 Text feature [remaining] present in test data point [True]
47 Text feature [although] present in test data point [True]
56 Text feature [addition] present in test data point [True]
57 Text feature [specificity] present in test data point [True]
```

```
58 Text feature [database] present in test data point [True]
59 Text feature [type] present in test data point [True]
60 Text feature [include] present in test data point [True]
61 Text feature [function] present in test data point [True]
62 Text feature [table] present in test data point [True]
63 Text feature [discussion] present in test data point [True]
64 Text feature [predicted] present in test data point [True]
65 Text feature [structural] present in test data point [True]
66 Text feature [nih] present in test data point [True]
67 Text feature [missense] present in test data point [True]
68 Text feature [clear] present in test data point [True]
69 Text feature [provide] present in test data point [True]
71 Text feature [possible] present in test data point [True]
72 Text feature [wild] present in test data point [True]
73 Text feature [set] present in test data point [True]
74 Text feature [use] present in test data point [True]
75 Text feature [genetic] present in test data point [True]
76 Text feature [protein] present in test data point [True]
77 Text feature [example] present in test data point [True]
78 Text feature [org] present in test data point [True]
79 Text feature [therefore] present in test data point [True]
80 Text feature [classification] present in test data point [True]
81 Text feature [affect] present in test data point [True]
82 Text feature [also] present in test data point [True]
83 Text feature [defined] present in test data point [True]
84 Text feature [three] present in test data point [True]
85 Text feature [note] present in test data point [True]
86 Text feature [system] present in test data point [True]
87 Text feature [activities] present in test data point [True]
88 Text feature [well] present in test data point [True]
89 Text feature [larger] present in test data point [True]
90 Text feature [cancer] present in test data point [True]
91 Text feature [analysis] present in test data point [True]
92 Text feature [several] present in test data point [True]
93 Text feature [one] present in test data point [True]
94 Text feature [additional] present in test data point [True]
95 Text feature [previously] present in test data point [True]
96 Text feature [least] present in test data point [True]
97 Text feature [corresponding] present in test data point [True]
98 Text feature [risk] present in test data point [True]
99 Text feature [however] present in test data point [True]
Out of the top  100  features  79 are present in query point
```

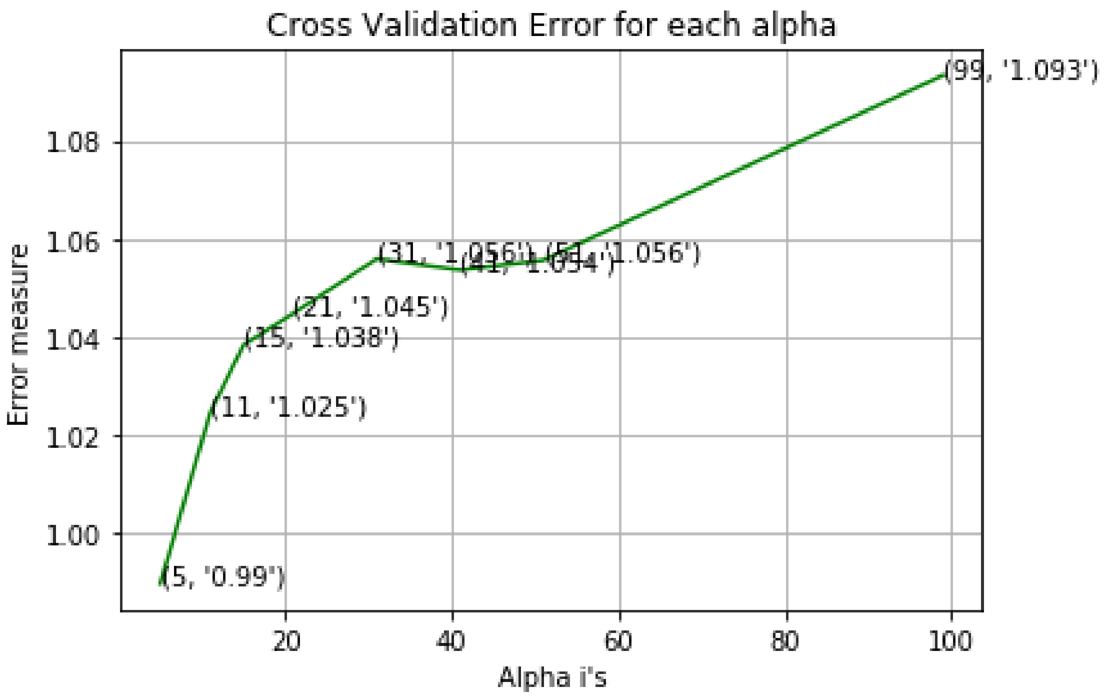
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```
In [84]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # -------------------------
```

```python
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, p
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 5
Log Loss : 0.9895501472998438
for alpha = 11
Log Loss : 1.0245204246143278
for alpha = 15
Log Loss : 1.0383149622735617
for alpha = 21
Log Loss : 1.0447780954092654
for alpha = 31
Log Loss : 1.0558945004354114
for alpha = 41
Log Loss : 1.0536163877393447
for alpha = 51
Log Loss : 1.0555972184759035
for alpha = 99
Log Loss : 1.0933268970097554
```



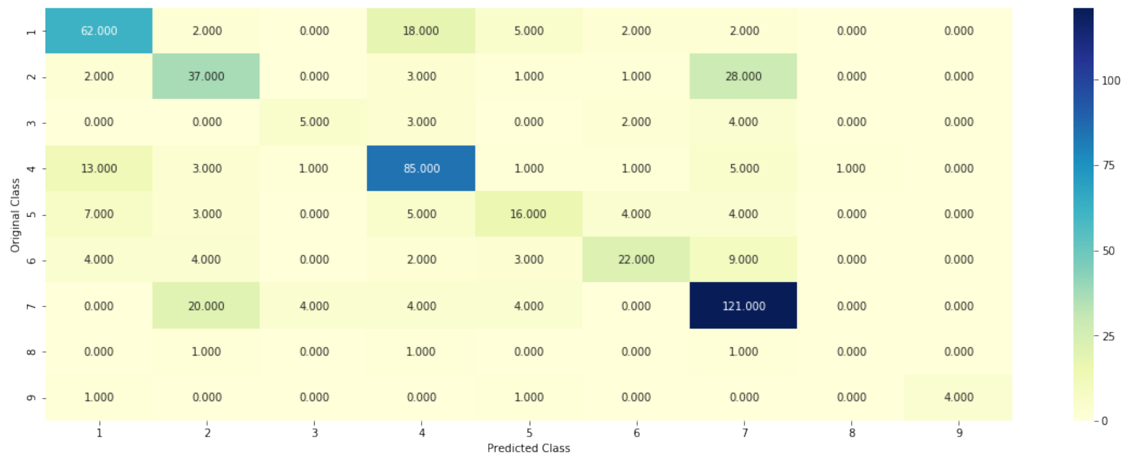
```
For values of best alpha =  5 The train log loss is: 0.4755616714256607
For values of best alpha =  5 The cross validation log loss is: 0.9895501472998438
For values of best alpha =  5 The test log loss is: 1.1052903577675415
```

4.2.2. Testing the model with best hyper paramters

```
In [85]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, 
         # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

         # methods of
         # fit(X, y) : Fit the model using X as training data and y as target values
         # predict(X):Predict the class labels for the provided data
         # predict_proba(X):Return probability estimates for the test data X.
         #-------------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
         #-------------------------------------
         clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodin
```

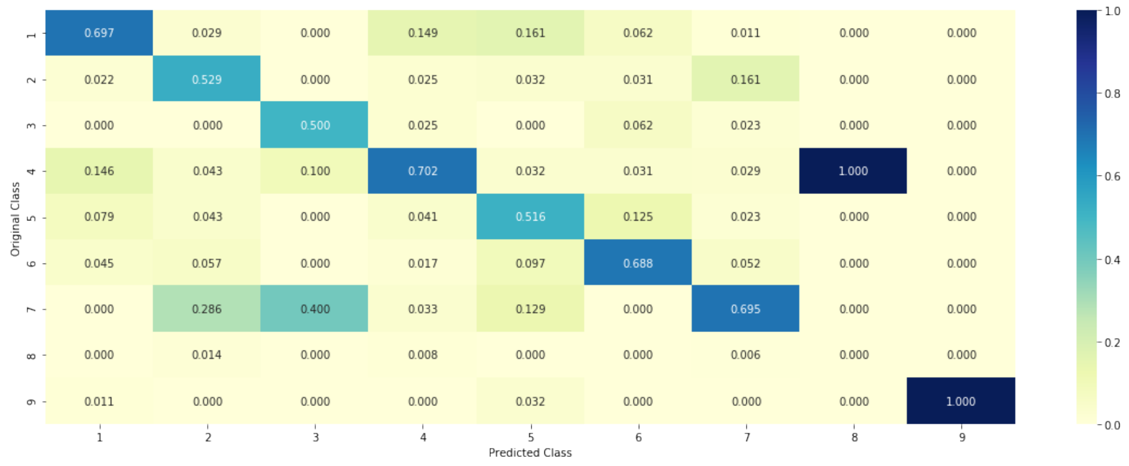
```
Log loss : 0.9895501472998438
Number of mis-classified points : 0.3383458646616541
```

```
-------------------- Confusion matrix --------------------
```



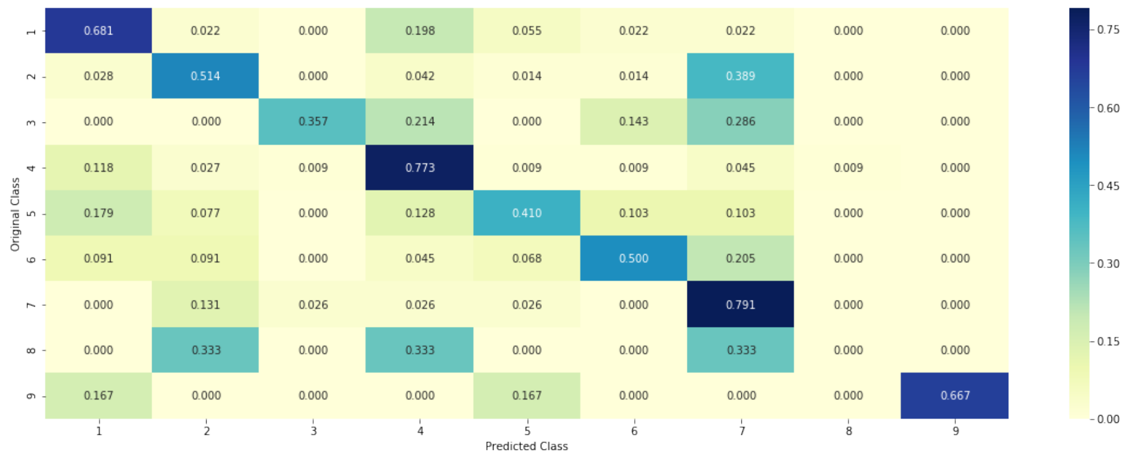
	1	2	3	4	5	6	7	8	9
1	62.000	2.000	0.000	18.000	5.000	2.000	2.000	0.000	0.000
2	2.000	37.000	0.000	3.000	1.000	1.000	28.000	0.000	0.000
3	0.000	0.000	5.000	3.000	0.000	2.000	4.000	0.000	0.000
4	13.000	3.000	1.000	85.000	1.000	1.000	5.000	1.000	0.000
5	7.000	3.000	0.000	5.000	16.000	4.000	4.000	0.000	0.000
6	4.000	4.000	0.000	2.000	3.000	22.000	9.000	0.000	0.000
7	0.000	20.000	4.000	4.000	4.000	0.000	121.000	0.000	0.000
8	0.000	1.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
9	1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	4.000

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.697	0.029	0.000	0.149	0.161	0.062	0.011	0.000	0.000
2	0.022	0.529	0.000	0.025	0.032	0.031	0.161	0.000	0.000
3	0.000	0.000	0.500	0.025	0.000	0.062	0.023	0.000	0.000
4	0.146	0.043	0.100	0.702	0.032	0.031	0.029	1.000	0.000
5	0.079	0.043	0.000	0.041	0.516	0.125	0.023	0.000	0.000
6	0.045	0.057	0.000	0.017	0.097	0.688	0.052	0.000	0.000
7	0.000	0.286	0.400	0.033	0.129	0.000	0.695	0.000	0.000
8	0.000	0.014	0.000	0.008	0.000	0.000	0.006	0.000	0.000
9	0.011	0.000	0.000	0.000	0.032	0.000	0.000	0.000	1.000

```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.2.3.Sample Query point -1

```
In [86]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 1
         predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
         print("Predicted Class :", predicted_cls[0])
         print("Actual Class :", test_y[test_point_index])
         neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), al
         print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to clas
         print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 1
The  5  nearest neighbours of the test points belongs to classes [1 2 1 1 2]
Fequency of nearest points : Counter({1: 3, 2: 2})
```

4.2.4. Sample Query Point-2

```
In [87]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1)
         print("Predicted Class :", predicted_cls[0])
         print("Actual Class :", test_y[test_point_index])
```

```
        neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), al
        print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
        print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 5
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [1 1
Fequency of nearest points : Counter({1: 5})
```

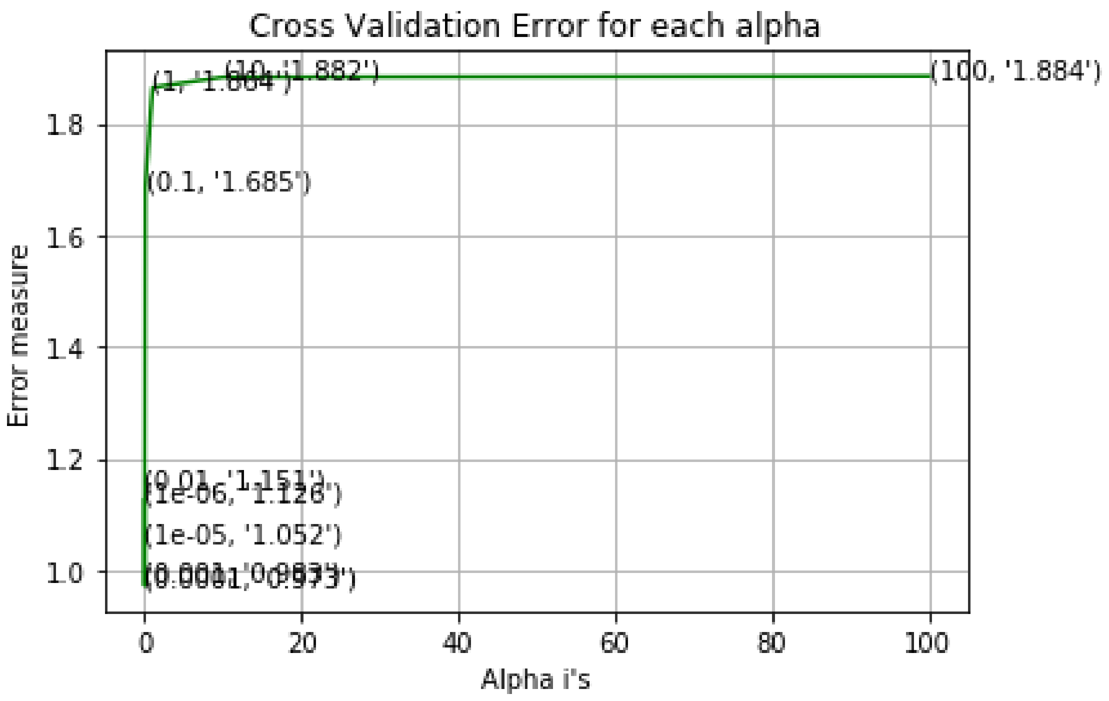
4.3. Logistic Regression
4.3.1. With Class balancing
4.3.1.1. Hyper paramter tuning

```
In [88]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # -------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
         #------------------------------



         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         #-------------------------------------
         # video link:
         #-------------------------------------

         alpha = [10 ** x for x in range(-6, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
```

```
        clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
        clf.fit(train_x_tfidf, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
        # to avoid rounding error while multiplying probabilites we use log-probability e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

    fig, ax = plt.subplots()
    ax.plot(alpha, cv_log_error_array,c='g')
    for i, txt in enumerate(np.round(cv_log_error_array,3)):
        ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
    plt.grid()
    plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
    plt.ylabel("Error measure")
    plt.show()


    best_alpha = np.argmin(cv_log_error_array)
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)

    predict_y = sig_clf.predict_proba(train_x_tfidf)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
    predict_y = sig_clf.predict_proba(cv_x_tfidf)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
    predict_y = sig_clf.predict_proba(test_x_tfidf)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-06
Log Loss : 1.1258211994294098
for alpha = 1e-05
Log Loss : 1.0520256421740386
for alpha = 0.0001
Log Loss : 0.9727344506894093
for alpha = 0.001
Log Loss : 0.9830599132063704
for alpha = 0.01
Log Loss : 1.1507220480535347
for alpha = 0.1
Log Loss : 1.6852744452227633
for alpha = 1
Log Loss : 1.8642063191513658
for alpha = 10
```

```
Log Loss : 1.8824355360292444
for alpha = 100
Log Loss : 1.8844369621590502
```



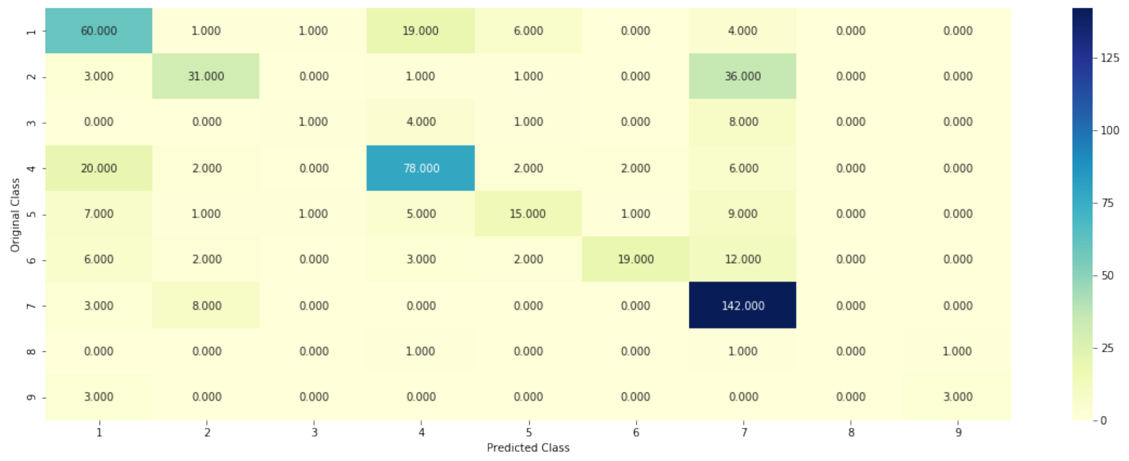
```
For values of best alpha =  0.0001 The train log loss is: 0.44707600670656245
For values of best alpha =  0.0001 The cross validation log loss is: 0.9727344506894093
For values of best alpha =  0.0001 The test log loss is: 1.0503413588586146
```

4.3.1.2. Testing the model with best hyper paramters

```
In [89]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
```

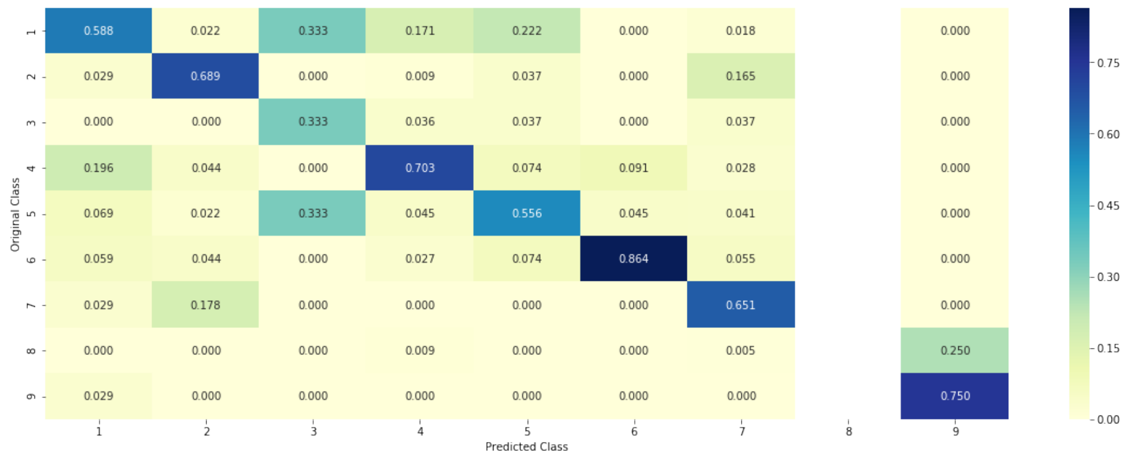
```
    #-------------------------------
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
    predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

```
Log loss : 0.9727344506894093
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```



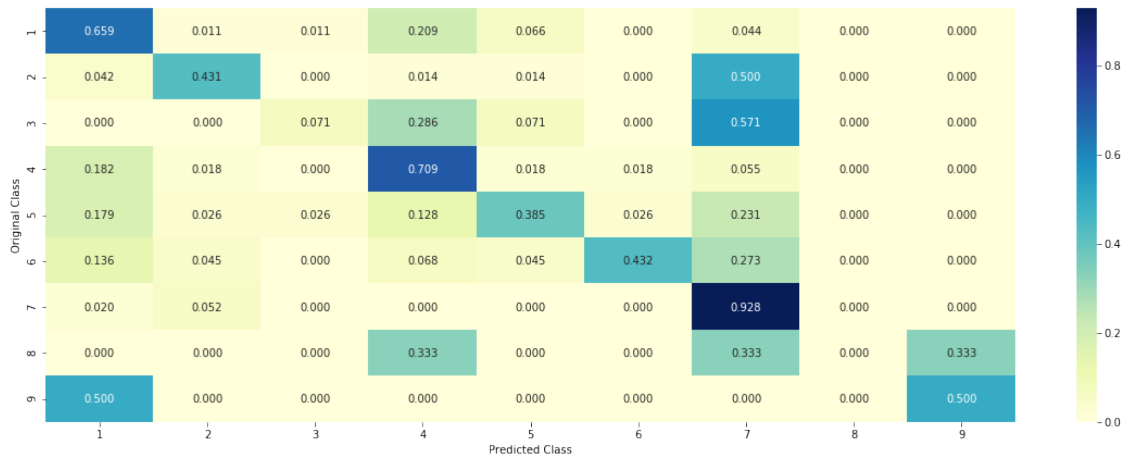
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	60.000	1.000	1.000	19.000	6.000	0.000	4.000	0.000	0.000
2	3.000	31.000	0.000	1.000	1.000	0.000	36.000	0.000	0.000
3	0.000	0.000	1.000	4.000	1.000	0.000	8.000	0.000	0.000
4	20.000	2.000	0.000	78.000	2.000	2.000	6.000	0.000	0.000
5	7.000	1.000	1.000	5.000	15.000	1.000	9.000	0.000	0.000
6	6.000	2.000	0.000	3.000	2.000	19.000	12.000	0.000	0.000
7	3.000	8.000	0.000	0.000	0.000	0.000	142.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	1.000
9	3.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.000

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.588	0.022	0.333	0.171	0.222	0.000	0.018		0.000
2	0.029	0.689	0.000	0.009	0.037	0.000	0.165		0.000
3	0.000	0.000	0.333	0.036	0.037	0.000	0.037		0.000
4	0.196	0.044	0.000	0.703	0.074	0.091	0.028		0.000
5	0.069	0.022	0.333	0.045	0.556	0.045	0.041		0.000
6	0.059	0.044	0.000	0.027	0.074	0.864	0.055		0.000
7	0.029	0.178	0.000	0.000	0.000	0.000	0.651		0.000
8	0.000	0.000	0.000	0.009	0.000	0.000	0.005		0.250
9	0.029	0.000	0.000	0.000	0.000	0.000	0.000		0.750

```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.1.3. Feature Importance

```
In [90]: def get_imp_feature_names(text, indices, removed_ind = []):
             word_present = 0
             tabulte_list = []
             incresingorder_ind = 0
             for i in indices:
                 if i < train_gene_feature_onehotCoding.shape[1]:
                     tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                 elif i< 18:
                     tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                 if ((i > 17) & (i not in removed_ind)) :
                     word = train_text_features[i]
                     yes_no = True if word in text.split() else False
                     if yes_no:
                         word_present += 1
                     tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
                 incresingorder_ind += 1
             print(word_present, "most importent features are present in our query point")
             print("-"*50)
             print("The features that are most importent of the ",predicted_cls[0]," class:")
             print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not'])
```

4.3.1.3.1. Correctly Classified point

```
In [93]: # from tabulate import tabulate
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         clf.fit(train_x_tfidf,train_y)
         test_point_index = 1
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
```

```
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.406  0.3322 0.0045 0.052  0.0186 0.0236 0.1575 0.0042 0.001
Actual Class : 1
--------------------------------------------------
161 Text feature [region] present in test data point [True]
218 Text feature [mutational] present in test data point [True]
248 Text feature [deletion] present in test data point [True]
262 Text feature [affect] present in test data point [True]
279 Text feature [insertion] present in test data point [True]
283 Text feature [across] present in test data point [True]
294 Text feature [loss] present in test data point [True]
297 Text feature [common] present in test data point [True]
309 Text feature [deletions] present in test data point [True]
321 Text feature [www] present in test data point [True]
325 Text feature [change] present in test data point [True]
330 Text feature [significantly] present in test data point [True]
341 Text feature [driven] present in test data point [True]
352 Text feature [somatic] present in test data point [True]
354 Text feature [conserved] present in test data point [True]
359 Text feature [interactions] present in test data point [True]
360 Text feature [binding] present in test data point [True]
361 Text feature [identify] present in test data point [True]
362 Text feature [17] present in test data point [True]
367 Text feature [grade] present in test data point [True]
374 Text feature [located] present in test data point [True]
375 Text feature [present] present in test data point [True]
378 Text feature [interact] present in test data point [True]
381 Text feature [one] present in test data point [True]
401 Text feature [ovarian] present in test data point [True]
408 Text feature [genome] present in test data point [True]
410 Text feature [subunit] present in test data point [True]
420 Text feature [sequenced] present in test data point [True]
422 Text feature [reverse] present in test data point [True]
426 Text feature [shows] present in test data point [True]
439 Text feature [role] present in test data point [True]
442 Text feature [sequencing] present in test data point [True]
448 Text feature [complex] present in test data point [True]
449 Text feature [fold] present in test data point [True]
453 Text feature [even] present in test data point [True]
454 Text feature [type] present in test data point [True]
457 Text feature [splice] present in test data point [True]
461 Text feature [46] present in test data point [True]
```

```
462 Text feature [large] present in test data point [True]
468 Text feature [co] present in test data point [True]
469 Text feature [next] present in test data point [True]
471 Text feature [colony] present in test data point [True]
476 Text feature [total] present in test data point [True]
479 Text feature [less] present in test data point [True]
480 Text feature [table] present in test data point [True]
485 Text feature [figures] present in test data point [True]
494 Text feature [terminal] present in test data point [True]
495 Text feature [results] present in test data point [True]
496 Text feature [3a] present in test data point [True]
Out of the top  500  features  49 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [94]: test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.3203 0.0047 0.001  0.1893 0.4712 0.0103 0.0012 0.0014 0.000
Actual Class : 5
--------------------------------------------------
126 Text feature [vitro] present in test data point [True]
150 Text feature [assessment] present in test data point [True]
155 Text feature [correlation] present in test data point [True]
156 Text feature [controls] present in test data point [True]
159 Text feature [studied] present in test data point [True]
161 Text feature [assays] present in test data point [True]
162 Text feature [neutral] present in test data point [True]
165 Text feature [rare] present in test data point [True]
168 Text feature [larger] present in test data point [True]
176 Text feature [assay] present in test data point [True]
180 Text feature [functional] present in test data point [True]
182 Text feature [pathogenic] present in test data point [True]
183 Text feature [manuscript] present in test data point [True]
189 Text feature [terminal] present in test data point [True]
191 Text feature [author] present in test data point [True]
192 Text feature [effect] present in test data point [True]
200 Text feature [brca2] present in test data point [True]
201 Text feature [clear] present in test data point [True]
```

```
203 Text feature [construct] present in test data point [True]
204 Text feature [variants] present in test data point [True]
206 Text feature [research] present in test data point [True]
208 Text feature [variant] present in test data point [True]
209 Text feature [classification] present in test data point [True]
210 Text feature [nih] present in test data point [True]
214 Text feature [small] present in test data point [True]
221 Text feature [phosphorylated] present in test data point [True]
223 Text feature [displayed] present in test data point [True]
225 Text feature [page] present in test data point [True]
228 Text feature [intermediate] present in test data point [True]
237 Text feature [support] present in test data point [True]
239 Text feature [class] present in test data point [True]
240 Text feature [example] present in test data point [True]
241 Text feature [number] present in test data point [True]
246 Text feature [note] present in test data point [True]
249 Text feature [addition] present in test data point [True]
251 Text feature [large] present in test data point [True]
257 Text feature [forms] present in test data point [True]
258 Text feature [calculated] present in test data point [True]
259 Text feature [s1] present in test data point [True]
261 Text feature [still] present in test data point [True]
263 Text feature [set] present in test data point [True]
270 Text feature [difference] present in test data point [True]
273 Text feature [assess] present in test data point [True]
274 Text feature [range] present in test data point [True]
275 Text feature [information] present in test data point [True]
277 Text feature [table] present in test data point [True]
282 Text feature [system] present in test data point [True]
285 Text feature [2010] present in test data point [True]
287 Text feature [described] present in test data point [True]
289 Text feature [database] present in test data point [True]
292 Text feature [known] present in test data point [True]
293 Text feature [include] present in test data point [True]
294 Text feature [targets] present in test data point [True]
295 Text feature [reported] present in test data point [True]
296 Text feature [stable] present in test data point [True]
298 Text feature [coding] present in test data point [True]
299 Text feature [experimental] present in test data point [True]
302 Text feature [even] present in test data point [True]
305 Text feature [type] present in test data point [True]
307 Text feature [criteria] present in test data point [True]
313 Text feature [damage] present in test data point [True]
314 Text feature [double] present in test data point [True]
316 Text feature [data] present in test data point [True]
317 Text feature [luciferase] present in test data point [True]
318 Text feature [transfected] present in test data point [True]
319 Text feature [transactivation] present in test data point [True]
```

```
320 Text feature [sequence] present in test data point [True]
322 Text feature [defects] present in test data point [True]
324 Text feature [seven] present in test data point [True]
329 Text feature [regions] present in test data point [True]
330 Text feature [prior] present in test data point [True]
332 Text feature [biochemical] present in test data point [True]
336 Text feature [provide] present in test data point [True]
338 Text feature [showed] present in test data point [True]
343 Text feature [corresponding] present in test data point [True]
344 Text feature [pocket] present in test data point [True]
345 Text feature [assessed] present in test data point [True]
346 Text feature [gene] present in test data point [True]
349 Text feature [wild] present in test data point [True]
351 Text feature [carrying] present in test data point [True]
352 Text feature [hours] present in test data point [True]
353 Text feature [effects] present in test data point [True]
362 Text feature [low] present in test data point [True]
363 Text feature [combined] present in test data point [True]
364 Text feature [ca] present in test data point [True]
366 Text feature [model] present in test data point [True]
367 Text feature [mammalian] present in test data point [True]
372 Text feature [evaluated] present in test data point [True]
375 Text feature [based] present in test data point [True]
376 Text feature [used] present in test data point [True]
380 Text feature [wt] present in test data point [True]
381 Text feature [found] present in test data point [True]
382 Text feature [sensitivity] present in test data point [True]
383 Text feature [structural] present in test data point [True]
385 Text feature [promoter] present in test data point [True]
386 Text feature [full] present in test data point [True]
388 Text feature [measured] present in test data point [True]
391 Text feature [possible] present in test data point [True]
393 Text feature [absence] present in test data point [True]
394 Text feature [containing] present in test data point [True]
395 Text feature [ability] present in test data point [True]
399 Text feature [suggested] present in test data point [True]
401 Text feature [downstream] present in test data point [True]
403 Text feature [published] present in test data point [True]
404 Text feature [genes] present in test data point [True]
405 Text feature [45] present in test data point [True]
409 Text feature [examined] present in test data point [True]
411 Text feature [region] present in test data point [True]
413 Text feature [specificity] present in test data point [True]
414 Text feature [screening] present in test data point [True]
415 Text feature [repair] present in test data point [True]
417 Text feature [would] present in test data point [True]
418 Text feature [yet] present in test data point [True]
419 Text feature [introduction] present in test data point [True]
```

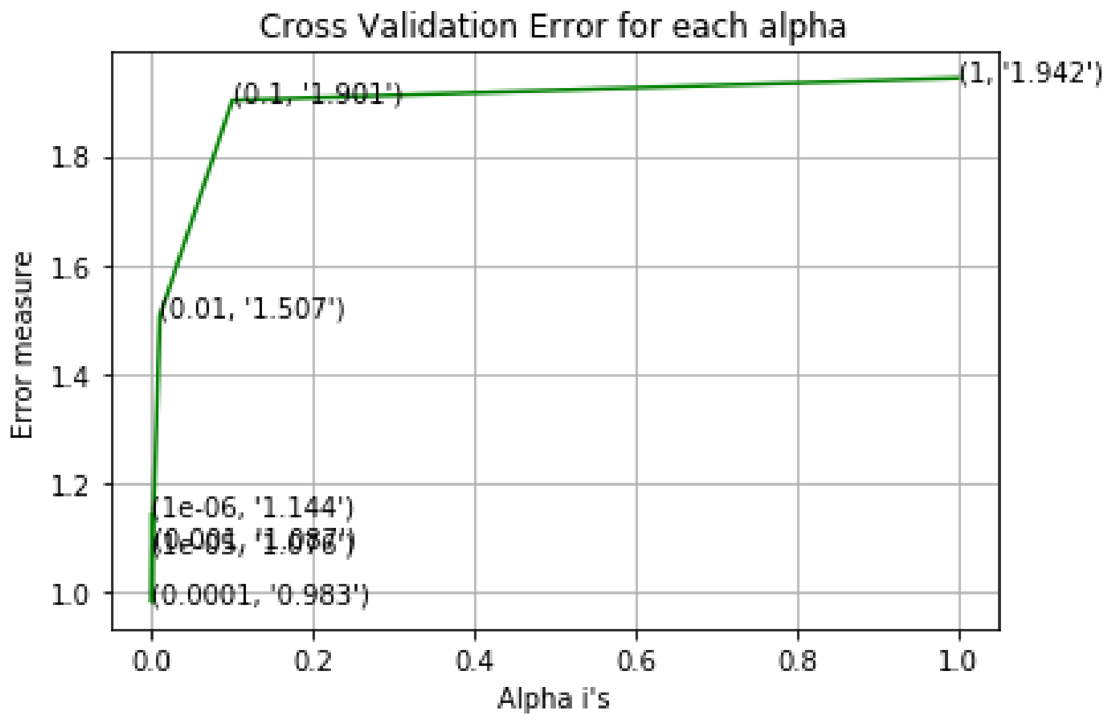
```
420 Text feature [method] present in test data point [True]
427 Text feature [remaining] present in test data point [True]
428 Text feature [length] present in test data point [True]
429 Text feature [discussion] present in test data point [True]
430 Text feature [included] present in test data point [True]
432 Text feature [single] present in test data point [True]
433 Text feature [well] present in test data point [True]
438 Text feature [results] present in test data point [True]
439 Text feature [80] present in test data point [True]
440 Text feature [tested] present in test data point [True]
443 Text feature [additional] present in test data point [True]
444 Text feature [transfection] present in test data point [True]
445 Text feature [however] present in test data point [True]
446 Text feature [transcription] present in test data point [True]
447 Text feature [44] present in test data point [True]
448 Text feature [2011] present in test data point [True]
449 Text feature [according] present in test data point [True]
450 Text feature [fact] present in test data point [True]
451 Text feature [genetic] present in test data point [True]
452 Text feature [org] present in test data point [True]
453 Text feature [methods] present in test data point [True]
454 Text feature [42] present in test data point [True]
456 Text feature [required] present in test data point [True]
459 Text feature [22] present in test data point [True]
462 Text feature [least] present in test data point [True]
465 Text feature [previously] present in test data point [True]
466 Text feature [represent] present in test data point [True]
467 Text feature [40] present in test data point [True]
470 Text feature [protein] present in test data point [True]
471 Text feature [vector] present in test data point [True]
473 Text feature [events] present in test data point [True]
474 Text feature [generation] present in test data point [True]
476 Text feature [deleterious] present in test data point [True]
480 Text feature [49] present in test data point [True]
482 Text feature [52] present in test data point [True]
487 Text feature [day] present in test data point [True]
488 Text feature [particular] present in test data point [True]
490 Text feature [established] present in test data point [True]
491 Text feature [linked] present in test data point [True]
492 Text feature [brct] present in test data point [True]
495 Text feature [relatively] present in test data point [True]
496 Text feature [target] present in test data point [True]
499 Text feature [analyses] present in test data point [True]
Out of the top  500  features  157 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```
In [95]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # -------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
         #------------------------------



         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         #-------------------------------------
         # video link:
         #-------------------------------------

         alpha = [10 ** x for x in range(-6, 1)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(train_x_tfidf, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidf, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

         fig, ax = plt.subplots()
         ax.plot(alpha, cv_log_error_array,c='g')
         for i, txt in enumerate(np.round(cv_log_error_array,3)):
             ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
         plt.grid()
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()


         best_alpha = np.argmin(cv_log_error_array)
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         clf.fit(train_x_tfidf, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_tfidf, train_y)

         predict_y = sig_clf.predict_proba(train_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
         predict_y = sig_clf.predict_proba(cv_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
         predict_y = sig_clf.predict_proba(test_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

```
for alpha = 1e-06
Log Loss : 1.1437266763626077
for alpha = 1e-05
Log Loss : 1.075584644635899
for alpha = 0.0001
Log Loss : 0.983237703525922
for alpha = 0.001
Log Loss : 1.0874329426786546
for alpha = 0.01
Log Loss : 1.50661050357483
for alpha = 0.1
Log Loss : 1.9012819354629742
for alpha = 1
Log Loss : 1.9419761919253125
```

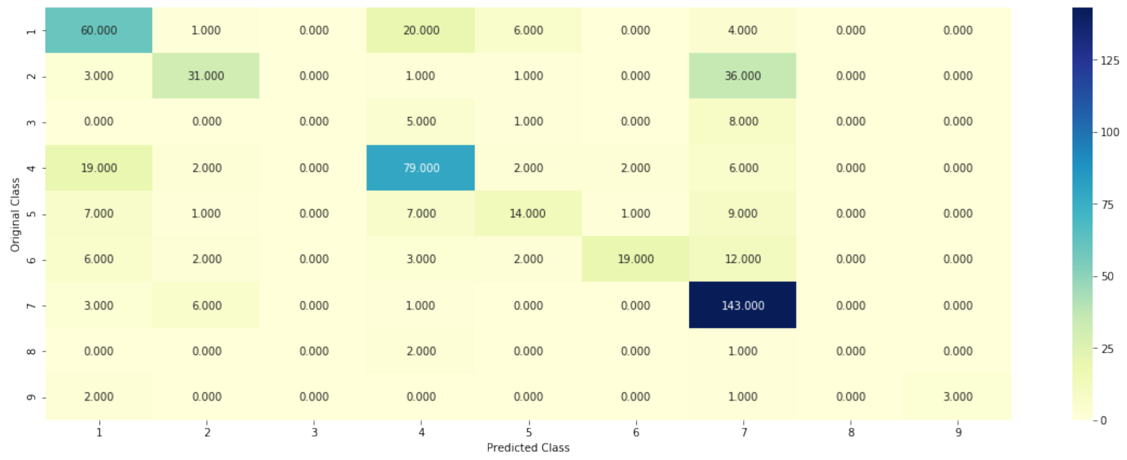


```
For values of best alpha =  0.0001 The train log loss is: 0.44313845948755093
For values of best alpha =  0.0001 The cross validation log loss is: 0.983237703525922
For values of best alpha =  0.0001 The test log loss is: 1.07080876593491
```

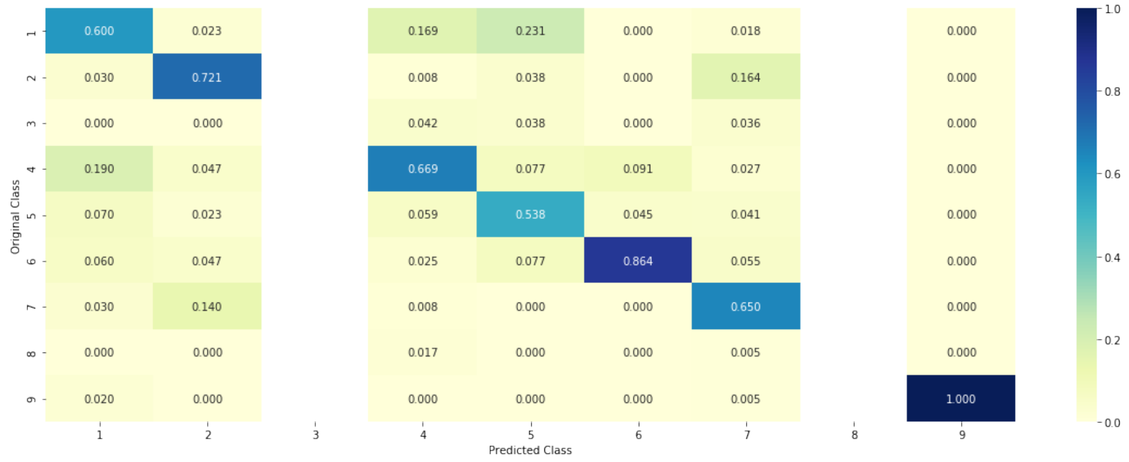
4.3.2.2. Testing model with best hyper parameters

```
In [96]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

         # some of methods
         # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
         # predict(X)        Predict class labels for samples in X.

         #-------------------------------
         # video link:
         #------------------------------

         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

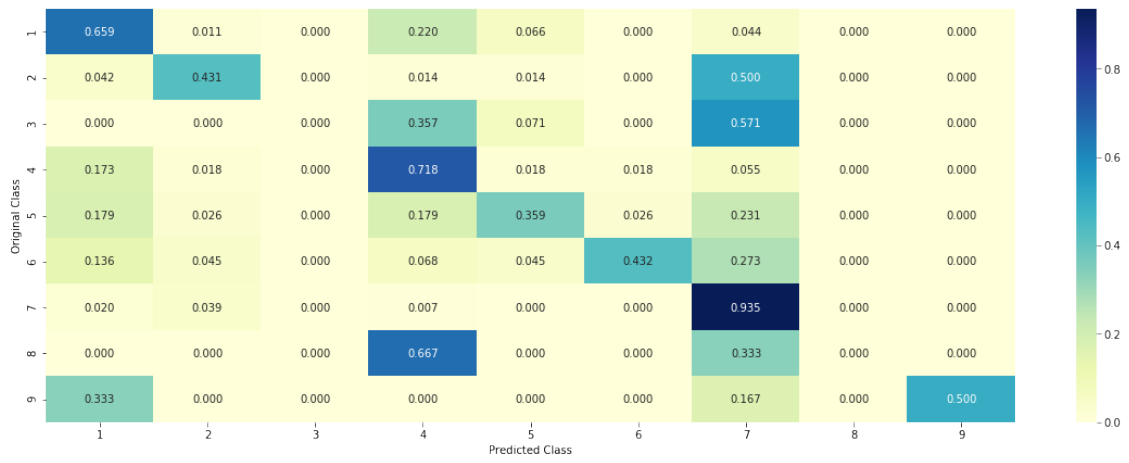
```
Log loss : 0.983237703525922
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.2.3. Feature Importance, Correctly Classified point

```
In [97]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         clf.fit(train_x_tfidf,train_y)
         test_point_index = 1
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
         print("-"*50)
         get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4035 0.3286 0.0075 0.0493 0.0182 0.023  0.1641 0.0039 0.001
Actual Class : 1
--------------------------------------------------
180 Text feature [region] present in test data point [True]
230 Text feature [mutational] present in test data point [True]
251 Text feature [deletion] present in test data point [True]
275 Text feature [affect] present in test data point [True]
276 Text feature [insertion] present in test data point [True]
284 Text feature [loss] present in test data point [True]
288 Text feature [across] present in test data point [True]
291 Text feature [common] present in test data point [True]
315 Text feature [deletions] present in test data point [True]
317 Text feature [www] present in test data point [True]
325 Text feature [significantly] present in test data point [True]
327 Text feature [change] present in test data point [True]
332 Text feature [somatic] present in test data point [True]
338 Text feature [driven] present in test data point [True]
```

```
347 Text feature [17] present in test data point [True]
351 Text feature [conserved] present in test data point [True]
356 Text feature [binding] present in test data point [True]
360 Text feature [identify] present in test data point [True]
366 Text feature [interactions] present in test data point [True]
374 Text feature [grade] present in test data point [True]
377 Text feature [located] present in test data point [True]
382 Text feature [one] present in test data point [True]
387 Text feature [present] present in test data point [True]
388 Text feature [interact] present in test data point [True]
399 Text feature [ovarian] present in test data point [True]
406 Text feature [genome] present in test data point [True]
412 Text feature [subunit] present in test data point [True]
416 Text feature [reverse] present in test data point [True]
420 Text feature [role] present in test data point [True]
421 Text feature [shows] present in test data point [True]
425 Text feature [fold] present in test data point [True]
427 Text feature [type] present in test data point [True]
430 Text feature [sequenced] present in test data point [True]
431 Text feature [sequencing] present in test data point [True]
434 Text feature [complex] present in test data point [True]
446 Text feature [46] present in test data point [True]
448 Text feature [compared] present in test data point [True]
456 Text feature [figures] present in test data point [True]
461 Text feature [next] present in test data point [True]
462 Text feature [even] present in test data point [True]
464 Text feature [colony] present in test data point [True]
470 Text feature [less] present in test data point [True]
473 Text feature [large] present in test data point [True]
478 Text feature [mutation] present in test data point [True]
479 Text feature [splice] present in test data point [True]
483 Text feature [3a] present in test data point [True]
487 Text feature [co] present in test data point [True]
490 Text feature [confirmed] present in test data point [True]
491 Text feature [table] present in test data point [True]
498 Text feature [total] present in test data point [True]
Out of the top  500  features  50 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [98]: test_point_index = 100
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[te
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 5
Predicted Class Probabilities: [[3.415e-01 5.000e-03 1.900e-03 2.215e-01 4.171e-01 9.600e-03 1
  1.400e-03 1.000e-04]]
Actual Class : 5
--------------------------------------------------
126 Text feature [vitro] present in test data point [True]
146 Text feature [correlation] present in test data point [True]
151 Text feature [assessment] present in test data point [True]
153 Text feature [controls] present in test data point [True]
161 Text feature [assays] present in test data point [True]
163 Text feature [studied] present in test data point [True]
164 Text feature [neutral] present in test data point [True]
167 Text feature [rare] present in test data point [True]
175 Text feature [larger] present in test data point [True]
180 Text feature [assay] present in test data point [True]
182 Text feature [functional] present in test data point [True]
183 Text feature [manuscript] present in test data point [True]
186 Text feature [pathogenic] present in test data point [True]
189 Text feature [terminal] present in test data point [True]
191 Text feature [author] present in test data point [True]
196 Text feature [effect] present in test data point [True]
199 Text feature [brca2] present in test data point [True]
200 Text feature [variants] present in test data point [True]
203 Text feature [construct] present in test data point [True]
204 Text feature [classification] present in test data point [True]
206 Text feature [nih] present in test data point [True]
208 Text feature [clear] present in test data point [True]
209 Text feature [variant] present in test data point [True]
213 Text feature [research] present in test data point [True]
219 Text feature [intermediate] present in test data point [True]
223 Text feature [small] present in test data point [True]
224 Text feature [page] present in test data point [True]
227 Text feature [phosphorylated] present in test data point [True]
228 Text feature [displayed] present in test data point [True]
233 Text feature [class] present in test data point [True]
236 Text feature [support] present in test data point [True]
241 Text feature [example] present in test data point [True]
243 Text feature [number] present in test data point [True]
244 Text feature [calculated] present in test data point [True]
247 Text feature [note] present in test data point [True]
250 Text feature [large] present in test data point [True]
252 Text feature [addition] present in test data point [True]
256 Text feature [set] present in test data point [True]
257 Text feature [difference] present in test data point [True]
260 Text feature [range] present in test data point [True]
```

```
268 Text feature [assess] present in test data point [True]
269 Text feature [information] present in test data point [True]
270 Text feature [s1] present in test data point [True]
272 Text feature [still] present in test data point [True]
281 Text feature [known] present in test data point [True]
283 Text feature [forms] present in test data point [True]
284 Text feature [table] present in test data point [True]
285 Text feature [experimental] present in test data point [True]
287 Text feature [database] present in test data point [True]
288 Text feature [system] present in test data point [True]
289 Text feature [include] present in test data point [True]
290 Text feature [stable] present in test data point [True]
291 Text feature [2010] present in test data point [True]
295 Text feature [targets] present in test data point [True]
297 Text feature [coding] present in test data point [True]
299 Text feature [type] present in test data point [True]
302 Text feature [criteria] present in test data point [True]
303 Text feature [reported] present in test data point [True]
305 Text feature [described] present in test data point [True]
312 Text feature [defects] present in test data point [True]
315 Text feature [pocket] present in test data point [True]
320 Text feature [even] present in test data point [True]
321 Text feature [seven] present in test data point [True]
323 Text feature [data] present in test data point [True]
325 Text feature [sequence] present in test data point [True]
326 Text feature [wild] present in test data point [True]
327 Text feature [transfected] present in test data point [True]
328 Text feature [prior] present in test data point [True]
329 Text feature [luciferase] present in test data point [True]
330 Text feature [transactivation] present in test data point [True]
331 Text feature [damage] present in test data point [True]
332 Text feature [regions] present in test data point [True]
339 Text feature [corresponding] present in test data point [True]
340 Text feature [sensitivity] present in test data point [True]
342 Text feature [provide] present in test data point [True]
343 Text feature [double] present in test data point [True]
344 Text feature [showed] present in test data point [True]
345 Text feature [biochemical] present in test data point [True]
346 Text feature [hours] present in test data point [True]
349 Text feature [structural] present in test data point [True]
350 Text feature [ca] present in test data point [True]
352 Text feature [combined] present in test data point [True]
353 Text feature [low] present in test data point [True]
356 Text feature [effects] present in test data point [True]
357 Text feature [based] present in test data point [True]
362 Text feature [screening] present in test data point [True]
363 Text feature [mammalian] present in test data point [True]
367 Text feature [assessed] present in test data point [True]
```

```
369 Text feature [used] present in test data point [True]
370 Text feature [gene] present in test data point [True]
373 Text feature [carrying] present in test data point [True]
374 Text feature [possible] present in test data point [True]
377 Text feature [specificity] present in test data point [True]
379 Text feature [ability] present in test data point [True]
381 Text feature [measured] present in test data point [True]
385 Text feature [method] present in test data point [True]
386 Text feature [full] present in test data point [True]
387 Text feature [promoter] present in test data point [True]
389 Text feature [model] present in test data point [True]
392 Text feature [containing] present in test data point [True]
393 Text feature [remaining] present in test data point [True]
394 Text feature [found] present in test data point [True]
395 Text feature [wt] present in test data point [True]
397 Text feature [methods] present in test data point [True]
398 Text feature [genes] present in test data point [True]
400 Text feature [evaluated] present in test data point [True]
405 Text feature [45] present in test data point [True]
411 Text feature [introduction] present in test data point [True]
412 Text feature [absence] present in test data point [True]
413 Text feature [yet] present in test data point [True]
415 Text feature [examined] present in test data point [True]
417 Text feature [suggested] present in test data point [True]
418 Text feature [genetic] present in test data point [True]
421 Text feature [downstream] present in test data point [True]
422 Text feature [region] present in test data point [True]
424 Text feature [published] present in test data point [True]
425 Text feature [discussion] present in test data point [True]
426 Text feature [80] present in test data point [True]
428 Text feature [would] present in test data point [True]
429 Text feature [included] present in test data point [True]
430 Text feature [repair] present in test data point [True]
433 Text feature [length] present in test data point [True]
434 Text feature [transfection] present in test data point [True]
436 Text feature [tested] present in test data point [True]
438 Text feature [fact] present in test data point [True]
439 Text feature [52] present in test data point [True]
440 Text feature [2011] present in test data point [True]
442 Text feature [transcription] present in test data point [True]
443 Text feature [single] present in test data point [True]
447 Text feature [42] present in test data point [True]
448 Text feature [org] present in test data point [True]
449 Text feature [well] present in test data point [True]
450 Text feature [however] present in test data point [True]
451 Text feature [results] present in test data point [True]
452 Text feature [least] present in test data point [True]
454 Text feature [according] present in test data point [True]
```

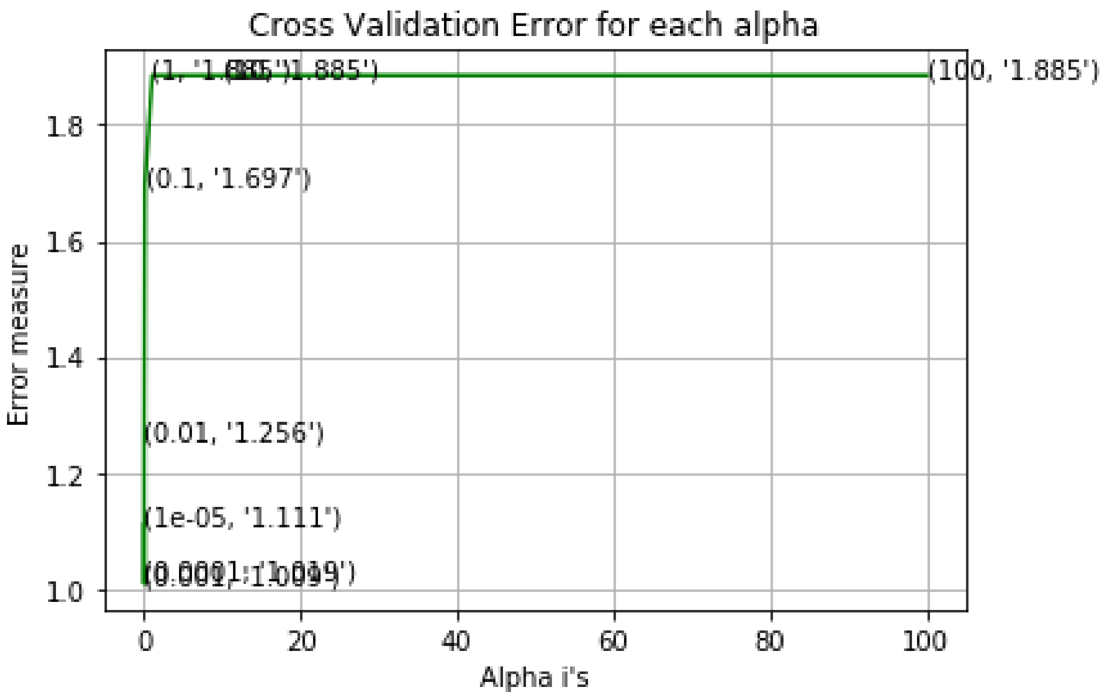
```
458 Text feature [additional] present in test data point [True]
462 Text feature [risk] present in test data point [True]
463 Text feature [deleterious] present in test data point [True]
464 Text feature [49] present in test data point [True]
467 Text feature [represent] present in test data point [True]
470 Text feature [44] present in test data point [True]
472 Text feature [protein] present in test data point [True]
475 Text feature [22] present in test data point [True]
476 Text feature [previously] present in test data point [True]
478 Text feature [brct] present in test data point [True]
481 Text feature [day] present in test data point [True]
482 Text feature [40] present in test data point [True]
483 Text feature [vector] present in test data point [True]
484 Text feature [relatively] present in test data point [True]
487 Text feature [generation] present in test data point [True]
488 Text feature [events] present in test data point [True]
489 Text feature [particular] present in test data point [True]
490 Text feature [required] present in test data point [True]
492 Text feature [established] present in test data point [True]
Out of the top  500  features  155 are present in query point
```

4.4. Linear Support Vector Machines
4.4.1. Hyper paramter tuning

```
In [99]: # read more about support vector machines with linear kernals here http://scikit-lear

         # --------------------------------
         # default parameters
         # SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True, probability
         # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_sh

         # Some of methods of SVM()
         # fit(X, y, [sample_weight])        Fit the SVM model according to the given training
         # predict(X)        Perform classification on samples in X.
         # --------------------------------
         # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
         # --------------------------------




         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # some of the methods of CalibratedClassifierCV()
         # fit(X, y[, sample_weight])        Fit the calibrated model
         # get_params([deep])        Get parameters for this estimator.
         # predict(X)        Predict the target of new samples.
         # predict_proba(X)        Posterior probabilities of classification
         #-------------------------------------
         # video link:
         #-------------------------------------

         alpha = [10 ** x for x in range(-5, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for C =", i)
         #     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
             clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge'
             clf.fit(train_x_tfidf, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidf, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

         fig, ax = plt.subplots()
         ax.plot(alpha, cv_log_error_array,c='g')
         for i, txt in enumerate(np.round(cv_log_error_array,3)):
             ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
         plt.grid()
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()


         best_alpha = np.argmin(cv_log_error_array)
         # clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
         clf.fit(train_x_tfidf, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_tfidf, train_y)

         predict_y = sig_clf.predict_proba(train_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
         predict_y = sig_clf.predict_proba(cv_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
         predict_y = sig_clf.predict_proba(test_x_tfidf)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

```
for C = 1e-05
Log Loss : 1.111415257127735
for C = 0.0001
```

```
Log Loss : 1.019487191674101
for C = 0.001
Log Loss : 1.0085986491526244
for C = 0.01
Log Loss : 1.2559967123898954
for C = 0.1
Log Loss : 1.6966202503562129
for C = 1
Log Loss : 1.884869798107891
for C = 10
Log Loss : 1.8848695168376812
for C = 100
Log Loss : 1.8848695515280116
```



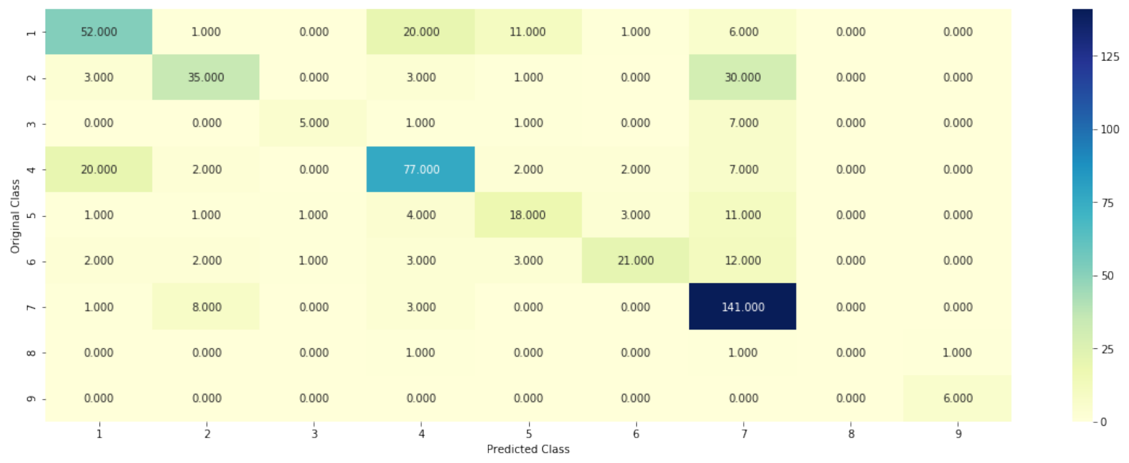
```
For values of best alpha =  0.001 The train log loss is: 0.5967498745530662
For values of best alpha =  0.001 The cross validation log loss is: 1.0085986491526244
For values of best alpha =  0.001 The test log loss is: 1.0964335626207993
```

4.4.2. Testing model with best hyper parameters

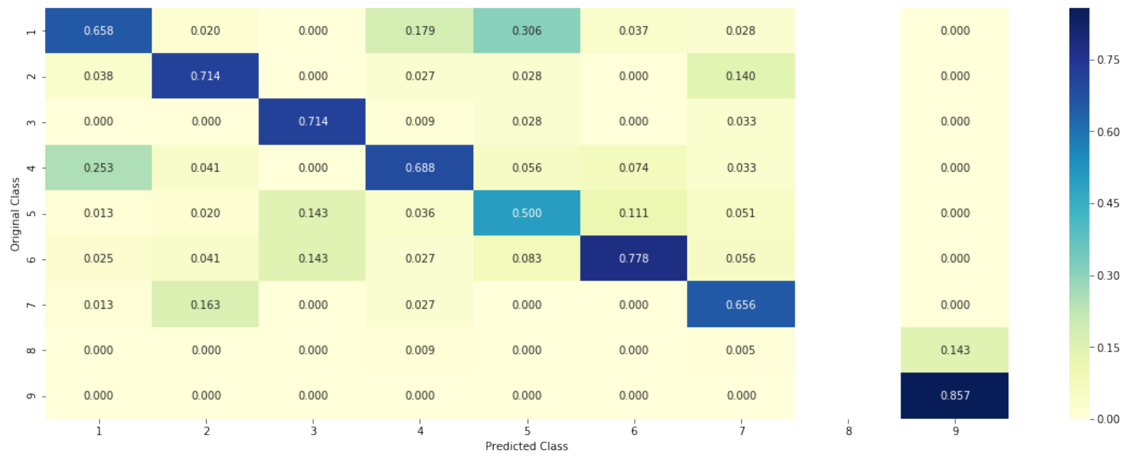
```
In [100]: # read more about support vector machines with linear kernals here http://scikit-lea

          # --------------------------------
```

```
    # default parameters
    # SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True, probabilit
    # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_s

    # Some of methods of SVM()
    # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
    # predict(X)        Perform classification on samples in X.
    # --------------------------------
    # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
    # --------------------------------


    # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='bala
    clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_stat
    predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

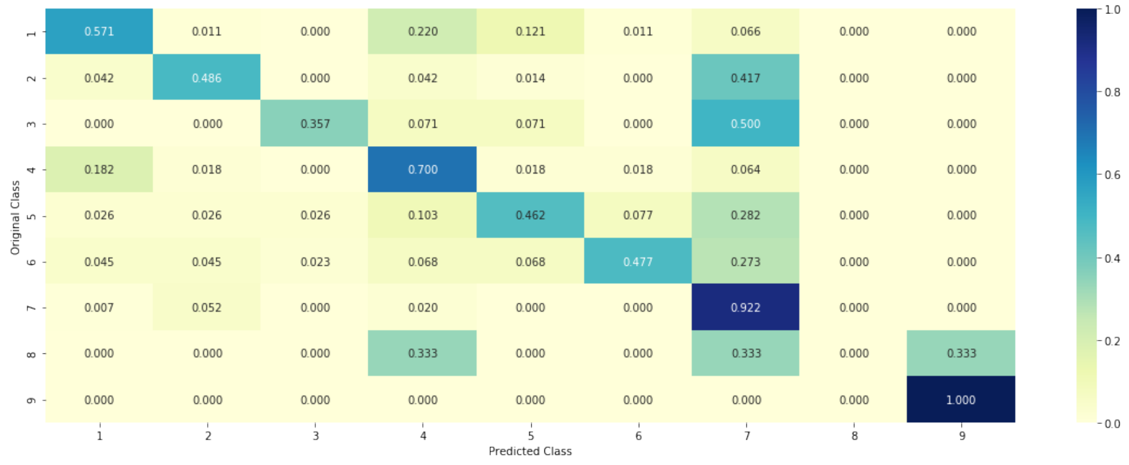
```
Log loss : 1.0085986491526244
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [101]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          clf.fit(train_x_tfidf,train_y)
          test_point_index = 1
          # test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[t
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen

Predicted Class : 1
Predicted Class Probabilities: [[0.3997 0.2821 0.0118 0.1001 0.0252 0.0352 0.1406 0.0035 0.001
Actual Class : 1
--------------------------------------------------
269 Text feature [across] present in test data point [True]
279 Text feature [common] present in test data point [True]
280 Text feature [insertion] present in test data point [True]
281 Text feature [region] present in test data point [True]
282 Text feature [mutational] present in test data point [True]
283 Text feature [significantly] present in test data point [True]
286 Text feature [subunit] present in test data point [True]
287 Text feature [driven] present in test data point [True]
289 Text feature [binding] present in test data point [True]
316 Text feature [affect] present in test data point [True]
323 Text feature [loss] present in test data point [True]
326 Text feature [somatic] present in test data point [True]
327 Text feature [identify] present in test data point [True]
330 Text feature [located] present in test data point [True]
333 Text feature [deletion] present in test data point [True]
341 Text feature [interactions] present in test data point [True]
344 Text feature [one] present in test data point [True]
347 Text feature [type] present in test data point [True]
354 Text feature [www] present in test data point [True]
355 Text feature [total] present in test data point [True]
356 Text feature [sequenced] present in test data point [True]
362 Text feature [ovarian] present in test data point [True]
372 Text feature [deletions] present in test data point [True]
406 Text feature [even] present in test data point [True]
418 Text feature [grade] present in test data point [True]
420 Text feature [next] present in test data point [True]
424 Text feature [conserved] present in test data point [True]
425 Text feature [17] present in test data point [True]
427 Text feature [interact] present in test data point [True]
431 Text feature [harboring] present in test data point [True]
436 Text feature [46] present in test data point [True]
437 Text feature [reverse] present in test data point [True]
438 Text feature [wild] present in test data point [True]
439 Text feature [gene] present in test data point [True]
442 Text feature [growth] present in test data point [True]
444 Text feature [sequencing] present in test data point [True]
453 Text feature [colony] present in test data point [True]
461 Text feature [confirmed] present in test data point [True]
464 Text feature [multiple] present in test data point [True]
465 Text feature [results] present in test data point [True]
466 Text feature [role] present in test data point [True]
```

```
467 Text feature [specific] present in test data point [True]
468 Text feature [change] present in test data point [True]
469 Text feature [less] present in test data point [True]
472 Text feature [present] present in test data point [True]
473 Text feature [stable] present in test data point [True]
477 Text feature [shows] present in test data point [True]
478 Text feature [3a] present in test data point [True]
479 Text feature [mutation] present in test data point [True]
481 Text feature [progression] present in test data point [True]
485 Text feature [fold] present in test data point [True]
487 Text feature [demonstrate] present in test data point [True]
492 Text feature [presence] present in test data point [True]
493 Text feature [tp53] present in test data point [True]
496 Text feature [controls] present in test data point [True]
499 Text feature [performed] present in test data point [True]
Out of the top  500  features  56 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [102]: test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[t
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1727 0.049  0.0066 0.1384 0.5761 0.0099 0.0421 0.0031 0.002
Actual Class : 5
--------------------------------------------------
4 Text feature [neutral] present in test data point [True]
145 Text feature [research] present in test data point [True]
163 Text feature [assays] present in test data point [True]
164 Text feature [clear] present in test data point [True]
165 Text feature [controls] present in test data point [True]
166 Text feature [vitro] present in test data point [True]
170 Text feature [functional] present in test data point [True]
171 Text feature [correlation] present in test data point [True]
173 Text feature [sensitivity] present in test data point [True]
174 Text feature [pathogenic] present in test data point [True]
175 Text feature [assay] present in test data point [True]
176 Text feature [org] present in test data point [True]
177 Text feature [studied] present in test data point [True]
178 Text feature [larger] present in test data point [True]
```

```
179 Text feature [intermediate] present in test data point [True]
181 Text feature [assessment] present in test data point [True]
183 Text feature [manuscript] present in test data point [True]
185 Text feature [construct] present in test data point [True]
193 Text feature [support] present in test data point [True]
199 Text feature [author] present in test data point [True]
201 Text feature [include] present in test data point [True]
203 Text feature [terminal] present in test data point [True]
207 Text feature [addition] present in test data point [True]
208 Text feature [rare] present in test data point [True]
209 Text feature [published] present in test data point [True]
212 Text feature [effect] present in test data point [True]
216 Text feature [variants] present in test data point [True]
217 Text feature [small] present in test data point [True]
219 Text feature [brca2] present in test data point [True]
225 Text feature [number] present in test data point [True]
226 Text feature [page] present in test data point [True]
227 Text feature [note] present in test data point [True]
230 Text feature [defects] present in test data point [True]
231 Text feature [difference] present in test data point [True]
235 Text feature [large] present in test data point [True]
236 Text feature [effects] present in test data point [True]
237 Text feature [nih] present in test data point [True]
242 Text feature [still] present in test data point [True]
244 Text feature [s1] present in test data point [True]
245 Text feature [classification] present in test data point [True]
246 Text feature [displayed] present in test data point [True]
251 Text feature [example] present in test data point [True]
252 Text feature [class] present in test data point [True]
253 Text feature [set] present in test data point [True]
255 Text feature [range] present in test data point [True]
256 Text feature [known] present in test data point [True]
257 Text feature [phosphorylated] present in test data point [True]
265 Text feature [regions] present in test data point [True]
267 Text feature [type] present in test data point [True]
268 Text feature [deleterious] present in test data point [True]
270 Text feature [system] present in test data point [True]
273 Text feature [targets] present in test data point [True]
282 Text feature [experimental] present in test data point [True]
284 Text feature [coding] present in test data point [True]
286 Text feature [wild] present in test data point [True]
291 Text feature [length] present in test data point [True]
293 Text feature [ca] present in test data point [True]
299 Text feature [sequence] present in test data point [True]
302 Text feature [hours] present in test data point [True]
305 Text feature [introduction] present in test data point [True]
306 Text feature [vector] present in test data point [True]
308 Text feature [full] present in test data point [True]
```

```
309 Text feature [transactivation] present in test data point [True]
310 Text feature [although] present in test data point [True]
311 Text feature [http] present in test data point [True]
316 Text feature [specificity] present in test data point [True]
318 Text feature [www] present in test data point [True]
320 Text feature [data] present in test data point [True]
323 Text feature [variant] present in test data point [True]
325 Text feature [low] present in test data point [True]
326 Text feature [genetic] present in test data point [True]
328 Text feature [prior] present in test data point [True]
329 Text feature [containing] present in test data point [True]
330 Text feature [possible] present in test data point [True]
333 Text feature [showed] present in test data point [True]
334 Text feature [assess] present in test data point [True]
335 Text feature [generation] present in test data point [True]
338 Text feature [examined] present in test data point [True]
339 Text feature [taken] present in test data point [True]
342 Text feature [26] present in test data point [True]
343 Text feature [database] present in test data point [True]
344 Text feature [based] present in test data point [True]
345 Text feature [activities] present in test data point [True]
346 Text feature [ability] present in test data point [True]
347 Text feature [information] present in test data point [True]
349 Text feature [luciferase] present in test data point [True]
350 Text feature [results] present in test data point [True]
354 Text feature [forms] present in test data point [True]
355 Text feature [described] present in test data point [True]
357 Text feature [transfected] present in test data point [True]
358 Text feature [fact] present in test data point [True]
359 Text feature [even] present in test data point [True]
360 Text feature [measured] present in test data point [True]
362 Text feature [stable] present in test data point [True]
363 Text feature [reported] present in test data point [True]
365 Text feature [wt] present in test data point [True]
366 Text feature [carrying] present in test data point [True]
368 Text feature [method] present in test data point [True]
371 Text feature [combined] present in test data point [True]
372 Text feature [promoter] present in test data point [True]
374 Text feature [exhibited] present in test data point [True]
375 Text feature [cell] present in test data point [True]
376 Text feature [table] present in test data point [True]
378 Text feature [included] present in test data point [True]
383 Text feature [calculated] present in test data point [True]
384 Text feature [directly] present in test data point [True]
386 Text feature [pocket] present in test data point [True]
387 Text feature [screening] present in test data point [True]
389 Text feature [genes] present in test data point [True]
390 Text feature [present] present in test data point [True]
```

```
392 Text feature [damage] present in test data point [True]
395 Text feature [values] present in test data point [True]
396 Text feature [experiments] present in test data point [True]
397 Text feature [gene] present in test data point [True]
399 Text feature [day] present in test data point [True]
401 Text feature [target] present in test data point [True]
402 Text feature [required] present in test data point [True]
403 Text feature [45] present in test data point [True]
407 Text feature [structural] present in test data point [True]
413 Text feature [according] present in test data point [True]
415 Text feature [used] present in test data point [True]
416 Text feature [evaluated] present in test data point [True]
418 Text feature [affinity] present in test data point [True]
420 Text feature [provide] present in test data point [True]
421 Text feature [mammalian] present in test data point [True]
422 Text feature [well] present in test data point [True]
423 Text feature [additional] present in test data point [True]
429 Text feature [criteria] present in test data point [True]
431 Text feature [model] present in test data point [True]
432 Text feature [driven] present in test data point [True]
433 Text feature [peptide] present in test data point [True]
434 Text feature [found] present in test data point [True]
435 Text feature [encoding] present in test data point [True]
437 Text feature [assessed] present in test data point [True]
441 Text feature [another] present in test data point [True]
445 Text feature [expression] present in test data point [True]
446 Text feature [state] present in test data point [True]
448 Text feature [use] present in test data point [True]
449 Text feature [2010] present in test data point [True]
450 Text feature [downstream] present in test data point [True]
464 Text feature [discussion] present in test data point [True]
469 Text feature [splicing] present in test data point [True]
470 Text feature [however] present in test data point [True]
472 Text feature [would] present in test data point [True]
474 Text feature [2011] present in test data point [True]
476 Text feature [analyses] present in test data point [True]
481 Text feature [s2] present in test data point [True]
482 Text feature [supplementary] present in test data point [True]
483 Text feature [comparison] present in test data point [True]
486 Text feature [analysis] present in test data point [True]
488 Text feature [side] present in test data point [True]
489 Text feature [series] present in test data point [True]
491 Text feature [80] present in test data point [True]
492 Text feature [biochemical] present in test data point [True]
493 Text feature [events] present in test data point [True]
494 Text feature [methods] present in test data point [True]
496 Text feature [41] present in test data point [True]
497 Text feature [sufficient] present in test data point [True]
```

```
Out of the top  500  features  158 are present in query point
```

4.5 Random Forest Classifier
4.5.1. Hyper paramter tuning (With One hot Encoding)

```
In [103]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # predict_proba (X)        Perform classification on samples in X.

          # some of attributes of  RandomForestClassifier()
          # feature_importances_ : array of shape = [n_features]
          # The feature importances (the higher, the more important the feature).

          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------


          # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
          # ----------------------------
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv
          #
          # some of the methods of CalibratedClassifierCV()
          # fit(X, y[, sample_weight])        Fit the calibrated model
          # get_params([deep])        Get parameters for this estimator.
          # predict(X)        Predict the target of new samples.
          # predict_proba(X)        Posterior probabilities of classification
          #-------------------------------------
          # video link:
          #-------------------------------------

          alpha = [100,200,500,1000,2000]
          max_depth = [5, 10]
          cv_log_error_array = []
          for i in alpha:
              for j in max_depth:
                  print("for n_estimators =", i,"and max depth = ", j)
                  clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, 
                  clf.fit(train_x_tfidf, train_y)
                  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                  sig_clf.fit(train_x_tfidf, train_y)
                  sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
                  cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
                  print("Log Loss :",log_loss(cv_y, sig_clf_probs))

          '''fig, ax = plt.subplots()
          features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
          ax.plot(features, cv_log_error_array,c='g')
          for i, txt in enumerate(np.round(cv_log_error_array,3)):
              ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_
          plt.grid()
          plt.title("Cross Validation Error for each alpha")
          plt.xlabel("Alpha i's")
          plt.ylabel("Error measure")
          plt.show()
          '''

          best_alpha = np.argmin(cv_log_error_array)
          clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
          clf.fit(train_x_tfidf, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_tfidf, train_y)

          predict_y = sig_clf.predict_proba(train_x_tfidf)
          print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log los
          predict_y = sig_clf.predict_proba(cv_x_tfidf)
          print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross valida
          predict_y = sig_clf.predict_proba(test_x_tfidf)
          print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
```

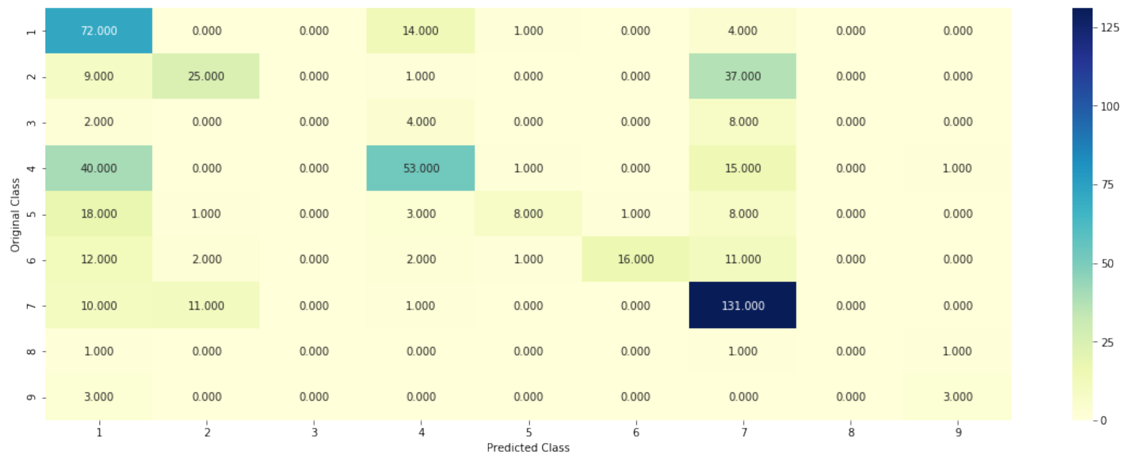
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1963536938648163
for n_estimators = 100 and max depth =  10
Log Loss : 1.2246365432223592
for n_estimators = 200 and max depth =  5
Log Loss : 1.1806100996022668
for n_estimators = 200 and max depth =  10
Log Loss : 1.2233734653075945
for n_estimators = 500 and max depth =  5
Log Loss : 1.170083796674727
for n_estimators = 500 and max depth =  10
Log Loss : 1.2238494368742707
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1703852990287391
for n_estimators = 1000 and max depth =  10
Log Loss : 1.219303583294475
```

```
for n_estimators = 2000 and max depth =  5
Log Loss : 1.169666437206652
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2173297493188506
For values of best estimator =  2000 The train log loss is: 0.8591261003242149
For values of best estimator =  2000 The cross validation log loss is: 1.169666437206652
For values of best estimator =  2000 The test log loss is: 1.216318891985087
```

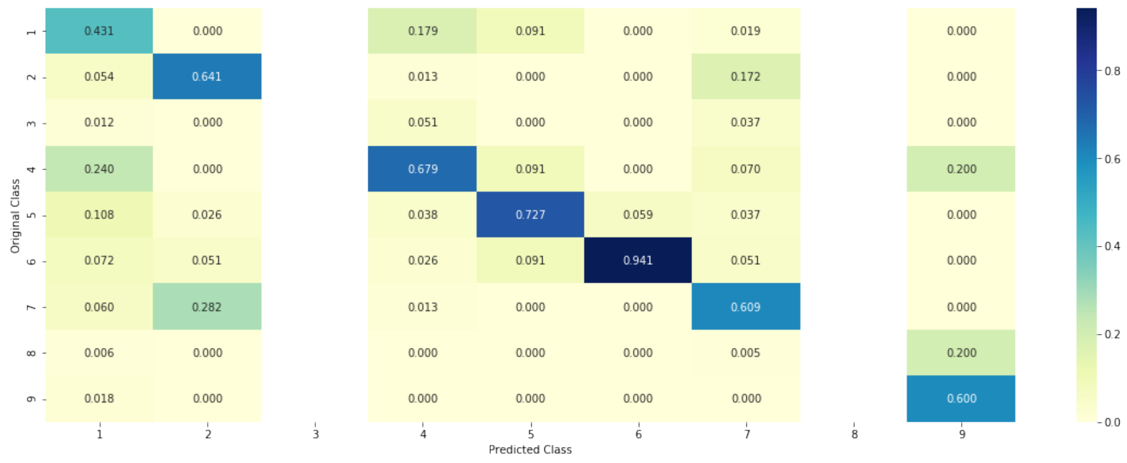
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

```
In [104]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # predict_proba (X)        Perform classification on samples in X.

          # some of attributes of  RandomForestClassifier()
          # feature_importances_ : array of shape = [n_features]
          # The feature importances (the higher, the more important the feature).

          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------

          clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
          predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

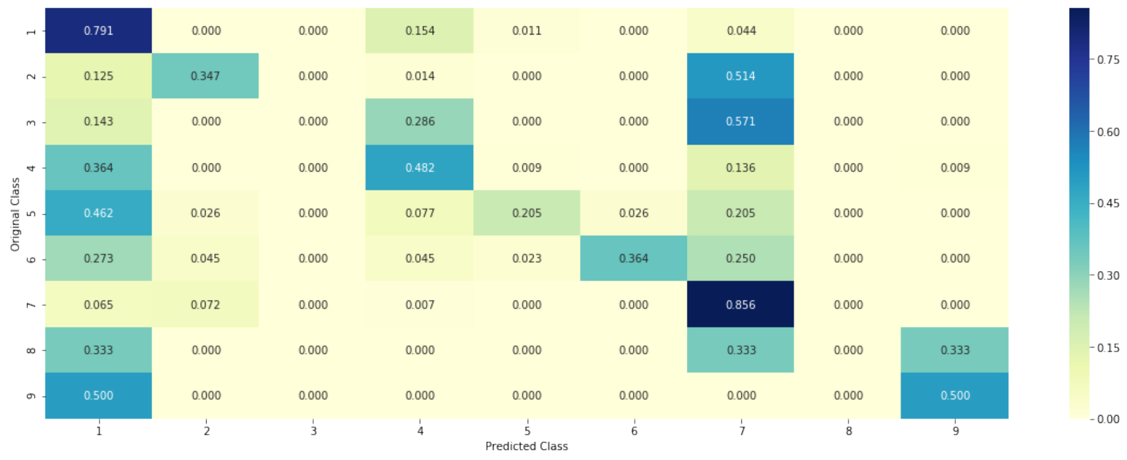
```
Log loss : 1.169666437206652
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```



-------------------- Precision matrix (Columm Sum=1) --------------------



-------------------- Recall matrix (Row sum=1) --------------------



4.5.3. Feature Importance
4.5.3.1. Correctly Classified point

```
In [105]: # test_point_index = 10
          clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
          clf.fit(train_x_tfidf, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_tfidf, train_y)

          test_point_index = 1
          no_feature = 100
          predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.feature_importances_)
          print("-"*50)
          get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],te
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.361  0.2189 0.018  0.1353 0.0607 0.0684 0.1157 0.009  0.013
Actual Class : 1
--------------------------------------------------
1 Text feature [kinase] present in test data point [True]
5 Text feature [activation] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
18 Text feature [loss] present in test data point [True]
19 Text feature [cells] present in test data point [True]
20 Text feature [deleterious] present in test data point [True]
21 Text feature [receptor] present in test data point [True]
22 Text feature [variants] present in test data point [True]
25 Text feature [protein] present in test data point [True]
```

```
28 Text feature [growth] present in test data point [True]
30 Text feature [cell] present in test data point [True]
32 Text feature [therapy] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [expression] present in test data point [True]
48 Text feature [signaling] present in test data point [True]
52 Text feature [inhibited] present in test data point [True]
54 Text feature [inhibition] present in test data point [True]
60 Text feature [pten] present in test data point [True]
68 Text feature [proteins] present in test data point [True]
71 Text feature [repair] present in test data point [True]
74 Text feature [resistance] present in test data point [True]
75 Text feature [lines] present in test data point [True]
77 Text feature [response] present in test data point [True]
87 Text feature [patients] present in test data point [True]
92 Text feature [ovarian] present in test data point [True]
95 Text feature [potential] present in test data point [True]
Out of the top  100  features  26 are present in query point
```

4.5.3.2. Inorrectly Classified point

```
In [106]: test_point_index = 100
          no_feature = 100
          predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[t
          print("Actuall Class :", test_y[test_point_index])
          indices = np.argsort(-clf.feature_importances_)
          print("-"*50)
          get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3659 0.0084 0.0157 0.1365 0.3651 0.0835 0.0191 0.004  0.0018
Actuall Class : 5
--------------------------------------------------
5 Text feature [activation] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [missense] present in test data point [True]
14 Text feature [function] present in test data point [True]
15 Text feature [constitutive] present in test data point [True]
18 Text feature [loss] present in test data point [True]
19 Text feature [cells] present in test data point [True]
20 Text feature [deleterious] present in test data point [True]
22 Text feature [variants] present in test data point [True]
25 Text feature [protein] present in test data point [True]
26 Text feature [brca1] present in test data point [True]
27 Text feature [classified] present in test data point [True]
```

```
29 Text feature [pathogenic] present in test data point [True]
30 Text feature [cell] present in test data point [True]
33 Text feature [neutral] present in test data point [True]
42 Text feature [expression] present in test data point [True]
45 Text feature [stability] present in test data point [True]
46 Text feature [functional] present in test data point [True]
48 Text feature [signaling] present in test data point [True]
51 Text feature [brca2] present in test data point [True]
53 Text feature [downstream] present in test data point [True]
64 Text feature [brct] present in test data point [True]
65 Text feature [variant] present in test data point [True]
68 Text feature [proteins] present in test data point [True]
69 Text feature [defective] present in test data point [True]
71 Text feature [repair] present in test data point [True]
77 Text feature [response] present in test data point [True]
79 Text feature [classification] present in test data point [True]
80 Text feature [sequence] present in test data point [True]
81 Text feature [yeast] present in test data point [True]
82 Text feature [ring] present in test data point [True]
85 Text feature [use] present in test data point [True]
86 Text feature [predicted] present in test data point [True]
87 Text feature [patients] present in test data point [True]
88 Text feature [expected] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
92 Text feature [ovarian] present in test data point [True]
93 Text feature [sensitivity] present in test data point [True]
95 Text feature [potential] present in test data point [True]
Out of the top  100  features  39 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

```
In [107]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # predict_proba (X)        Perform classification on samples in X.

          # some of attributes of  RandomForestClassifier()
          # feature_importances_ : array of shape = [n_features]
          # The feature importances (the higher, the more important the feature).
          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------


          # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
          # ----------------------------
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv
          #
          # some of the methods of CalibratedClassifierCV()
          # fit(X, y[, sample_weight])        Fit the calibrated model
          # get_params([deep])        Get parameters for this estimator.
          # predict(X)        Predict the target of new samples.
          # predict_proba(X)        Posterior probabilities of classification
          #-------------------------------------
          # video link:
          #-------------------------------------

          alpha = [10,50,100,200,500,1000]
          max_depth = [2,3,5,10]
          cv_log_error_array = []
          for i in alpha:
              for j in max_depth:
                  print("for n_estimators =", i,"and max depth = ", j)
                  clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, 
                  clf.fit(train_x_responseCoding, train_y)
                  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                  sig_clf.fit(train_x_responseCoding, train_y)
                  sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
                  cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
                  print("Log Loss :",log_loss(cv_y, sig_clf_probs))
          '''
          fig, ax = plt.subplots()
          features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
          ax.plot(features, cv_log_error_array,c='g')
          for i, txt in enumerate(np.round(cv_log_error_array,3)):
              ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_
          plt.grid()
          plt.title("Cross Validation Error for each alpha")
          plt.xlabel("Alpha i's")
          plt.ylabel("Error measure")
          plt.show()
          '''

          best_alpha = np.argmin(cv_log_error_array)
          clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini'
          clf.fit(train_x_responseCoding, train_y)
```

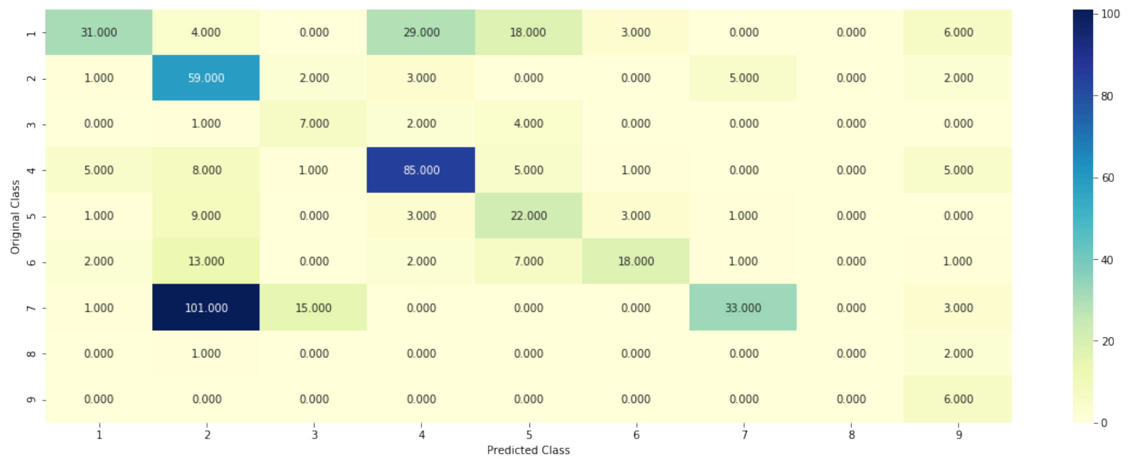
```
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_responseCoding, train_y)

          predict_y = sig_clf.predict_proba(train_x_responseCoding)
          print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is
          predict_y = sig_clf.predict_proba(cv_x_responseCoding)
          print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation
          predict_y = sig_clf.predict_proba(test_x_responseCoding)
          print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1108156464272225
for n_estimators = 10 and max depth =  3
Log Loss : 1.9624923579622622
for n_estimators = 10 and max depth =  5
Log Loss : 1.6054957634051872
for n_estimators = 10 and max depth =  10
Log Loss : 1.918312570479128
for n_estimators = 50 and max depth =  2
Log Loss : 1.79403859096244
for n_estimators = 50 and max depth =  3
Log Loss : 1.5891787208782076
for n_estimators = 50 and max depth =  5
Log Loss : 1.4272798048442568
for n_estimators = 50 and max depth =  10
Log Loss : 1.6617939377840498
for n_estimators = 100 and max depth =  2
Log Loss : 1.6274597630490775
for n_estimators = 100 and max depth =  3
Log Loss : 1.5336222924067782
for n_estimators = 100 and max depth =  5
Log Loss : 1.3382570537068512
for n_estimators = 100 and max depth =  10
Log Loss : 1.670248349578277
for n_estimators = 200 and max depth =  2
Log Loss : 1.700404457642025
for n_estimators = 200 and max depth =  3
Log Loss : 1.5145294650197045
for n_estimators = 200 and max depth =  5
Log Loss : 1.3816671435286338
for n_estimators = 200 and max depth =  10
Log Loss : 1.6426079450937834
for n_estimators = 500 and max depth =  2
Log Loss : 1.750570101481647
for n_estimators = 500 and max depth =  3
Log Loss : 1.5455748529366289
for n_estimators = 500 and max depth =  5
Log Loss : 1.3774477990787843
```

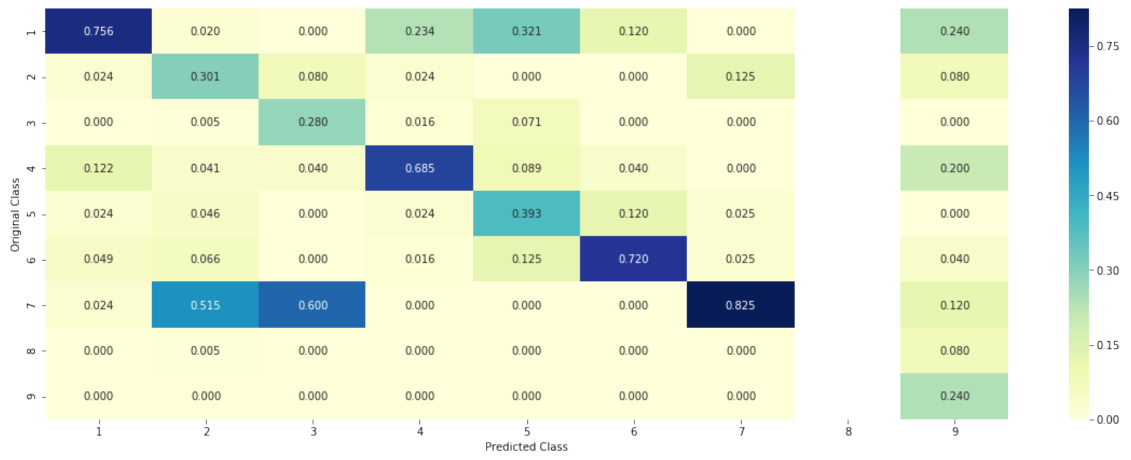
```
for n_estimators = 500 and max depth =  10
Log Loss : 1.684691522209715
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7064193184378014
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5719458933974577
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3670158325121695
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6699987071976068
For values of best alpha =  100 The train log loss is: 0.06623641922349537
For values of best alpha =  100 The cross validation log loss is: 1.3382570537068512
For values of best alpha =  100 The test log loss is: 1.4063519263971913
```

4.5.4. Testing model with best hyper parameters (Response Coding)

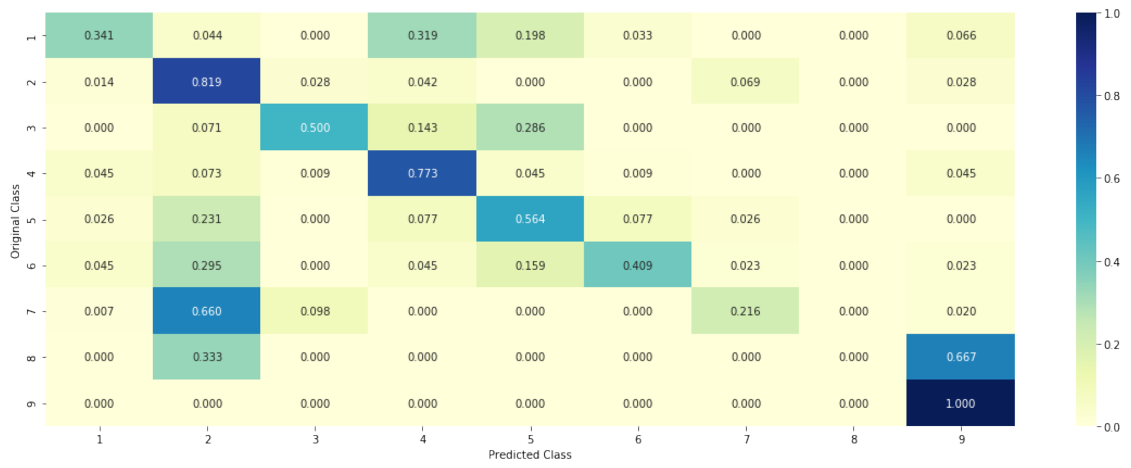
```
In [108]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # predict_proba (X)        Perform classification on samples in X.

          # some of attributes of  RandomForestClassifier()
          # feature_importances_ : array of shape = [n_features]
          # The feature importances (the higher, the more important the feature).

          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------

          clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=al
          predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCodin

Log loss : 1.3382570537068512
Number of mis-classified points : 0.5093984962406015
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.5.5. Feature Importance

4.5.5.1. Incorrectly Classified point

```
In [109]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini'
          clf.fit(train_x_responseCoding, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_responseCoding, train_y)


          test_point_index = 1
          no_feature = 27
          predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respons
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.feature_importances_)
          print("-"*50)
          for i in indices:
              if i<9:
                  print("Gene is important feature")
              elif i<18:
                  print("Variation is important feature")
              else:
                  print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0593 0.5352 0.0848 0.035  0.0398 0.0579 0.0914 0.0464 0.050
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Correctly Classified point

```
In [110]: test_point_index = 100
          predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respons
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.feature_importances_)
          print("-"*50)
          for i in indices:
              if i<9:
                  print("Gene is important feature")
              elif i<18:
                  print("Variation is important feature")
              else:
                  print("Text is important feature")
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0467 0.0052 0.0552 0.0571 0.6988 0.1255 0.0028 0.0039 0.004
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models
4.7.1 testing with hyper parameter tuning

```
In [111]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
          # class_weight=None, warm_start=False, average=False, n_iter=None)

          # some of methods
          # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic G
          # predict(X)        Predict class labels for samples in X.

          #-------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          #------------------------------



          # read more about support vector machines with linear kernals here http://scikit-lea
          # --------------------------------
          # default parameters
          # SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True, probabilit
          # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_s

          # Some of methods of SVM()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------


          # read more about support vector machines with linear kernals here http://scikit-lea
          # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # predict(X)        Perform classification on samples in X.
          # predict_proba (X)        Perform classification on samples in X.

          # some of attributes of  RandomForestClassifier()
          # feature_importances_ : array of shape = [n_features]
          # The feature importances (the higher, the more important the feature).

          # --------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          # --------------------------------


          clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced',
          clf1.fit(train_x_tfidf, train_y)
          sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

          clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', r
          clf2.fit(train_x_tfidf, train_y)
          sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


          clf3 = MultinomialNB(alpha=0.001)
          clf3.fit(train_x_tfidf, train_y)
          sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

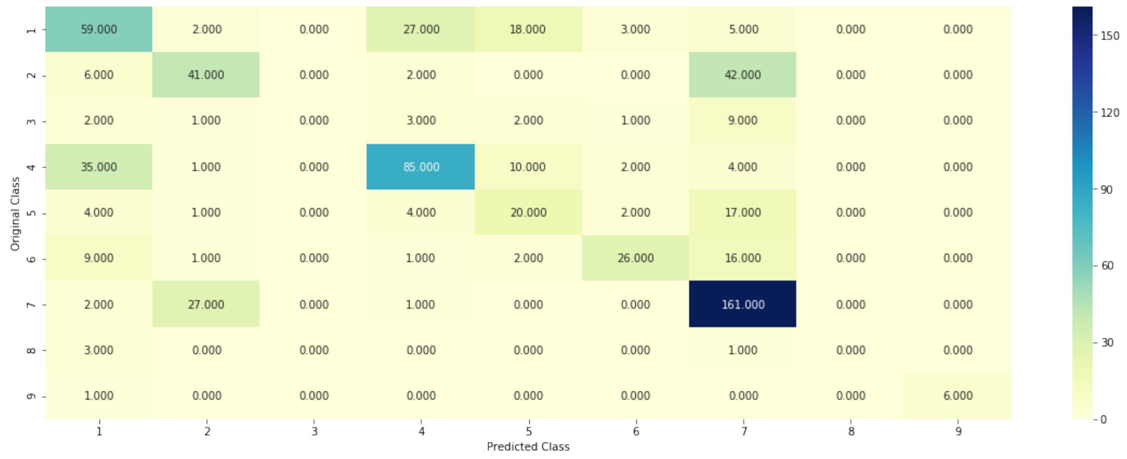
```
          sig_clf1.fit(train_x_tfidf, train_y)
          print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_pro
          sig_clf2.fit(train_x_tfidf, train_y)
          print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_
          sig_clf3.fit(train_x_tfidf, train_y)
          print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_
          print("-"*50)
          alpha = [0.0001,0.001,0.01,0.1,1,10]
          best_alpha = 999
          for i in alpha:
              lr = LogisticRegression(C=i)
              sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
              sclf.fit(train_x_tfidf, train_y)
              print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log
              log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
              if best_alpha > log_error:
                  best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.98
Support vector machines : Log Loss: 1.88
Naive Bayes : Log Loss: 1.14
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.029
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.478
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.105
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.260
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.656
```

4.7.2 testing the model with the best hyper parameters

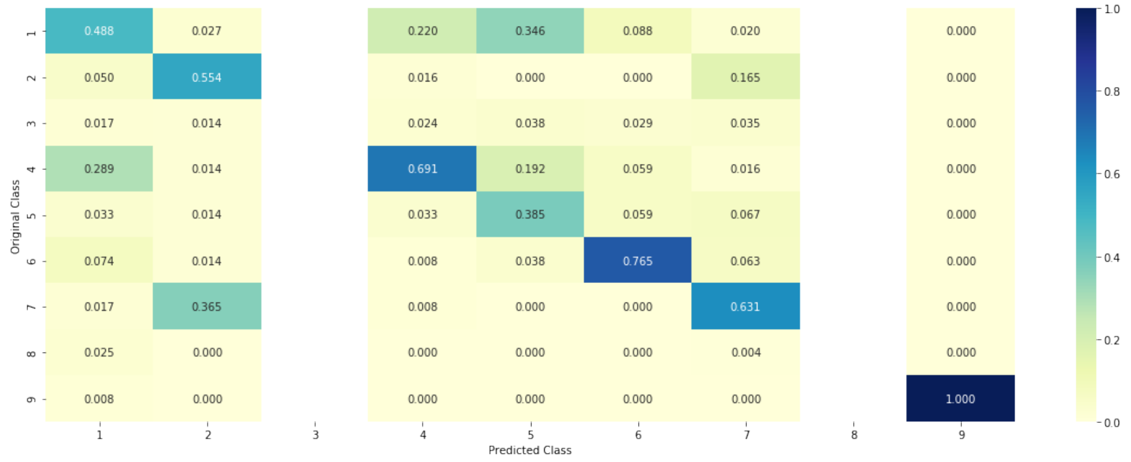
```
In [112]: lr = LogisticRegression(C=0.1)
          sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifie
          sclf.fit(train_x_tfidf, train_y)

          log_error = log_loss(train_y, sclf.predict_proba(train_x_tfidf))
          print("Log loss (train) on the stacking classifier :",log_error)

          log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
          print("Log loss (CV) on the stacking classifier :",log_error)

          log_error = log_loss(test_y, sclf.predict_proba(test_x_tfidf))
          print("Log loss (test) on the stacking classifier :",log_error)

          print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_tfid
          plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidf))
```

```
Log loss (train) on the stacking classifier : 0.5475895533229997
Log loss (CV) on the stacking classifier : 1.105263588849883
Log loss (test) on the stacking classifier : 1.223582955084984
Number of missclassified point : 0.40150375939849625
-------------------- Confusion matrix --------------------
```



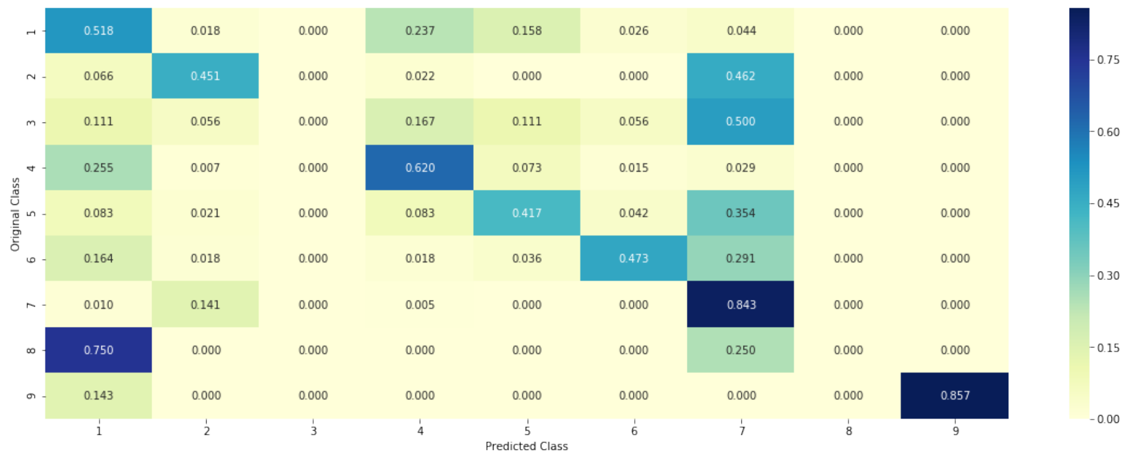
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	59.000	2.000	0.000	27.000	18.000	3.000	5.000	0.000	0.000
2	6.000	41.000	0.000	2.000	0.000	0.000	42.000	0.000	0.000
3	2.000	1.000	0.000	3.000	2.000	1.000	9.000	0.000	0.000
4	35.000	1.000	0.000	85.000	10.000	2.000	4.000	0.000	0.000
5	4.000	1.000	0.000	4.000	20.000	2.000	17.000	0.000	0.000
6	9.000	1.000	0.000	1.000	2.000	26.000	16.000	0.000	0.000
7	2.000	27.000	0.000	1.000	0.000	0.000	161.000	0.000	0.000
8	3.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.488	0.027		0.220	0.346	0.088	0.020		0.000
2	0.050	0.554		0.016	0.000	0.000	0.165		0.000
3	0.017	0.014		0.024	0.038	0.029	0.035		0.000
4	0.289	0.014		0.691	0.192	0.059	0.016		0.000
5	0.033	0.014		0.033	0.385	0.059	0.067		0.000
6	0.074	0.014		0.008	0.038	0.765	0.063		0.000
7	0.017	0.365		0.008	0.000	0.000	0.631		0.000
8	0.025	0.000		0.000	0.000	0.000	0.004		0.000
9	0.008	0.000		0.000	0.000	0.000	0.000		1.000

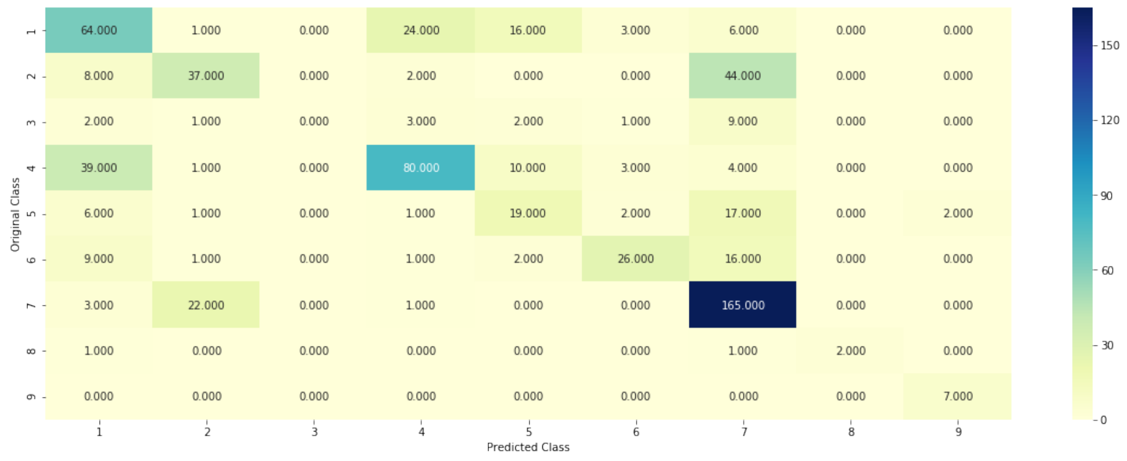
```
-------------------- Recall matrix (Row sum=1) --------------------
```



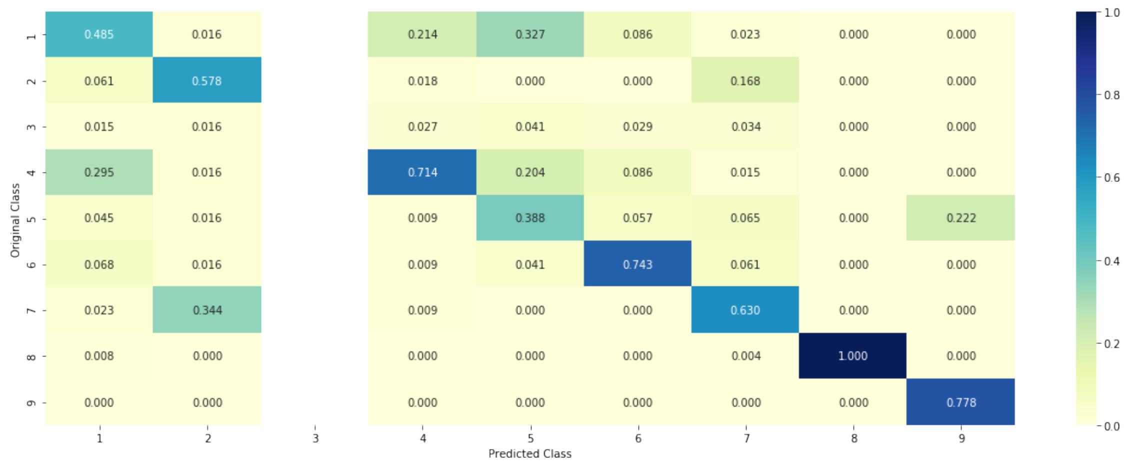
4.7.3 Maximum Voting classifier

```
In [113]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClass
          from sklearn.ensemble import VotingClassifier
          vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_
          vclf.fit(train_x_tfidf, train_y)
          print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_p
          print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(c
          print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_pro
          print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfid
          plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidf))
```

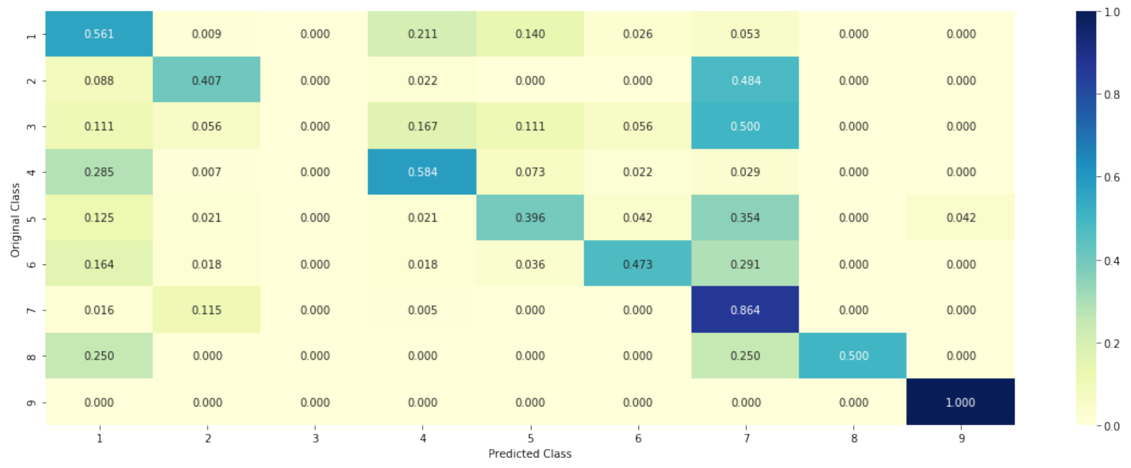
```
Log loss (train) on the VotingClassifier : 0.8338185510417913
Log loss (CV) on the VotingClassifier : 1.1658889773265164
Log loss (test) on the VotingClassifier : 1.232254762255751
Number of missclassified point : 0.39849624060150374
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



0.1 Conclusion:

```
In [114]: from prettytable import PrettyTable

          x = PrettyTable()
          x.field_names = ["Model", "Test Loss", "Misclassification"]
```

```
x.add_row(["Naive Bayes", "1.1367449679759856", "0.3533834586466165"])
x.add_row(["KNN", "0.9895501472998438", "0.3383458646616541"])
x.add_row(["Logistic Regression", "0.9727344506894093", "0.34398496240601506"])
x.add_row(["Linear SVM", "1.0085986491526244", "0.33270676691729323"])
x.add_row(["Random Forest", "1.169666437206652", "0.42105263157894735"])
x.add_row(["Naive Bayes", "1.232254762255751", "0.39849624060150374"])

print(x)
```

```
+---------------------+--------------------+---------------------+
|        Model        |     Test Loss      |  Misclassification  |
+---------------------+--------------------+---------------------+
|     Naive Bayes     | 1.1367449679759856 |  0.3533834586466165 |
|         KNN         | 0.9895501472998438 |  0.3383458646616541 |
| Logistic Regression | 0.9727344506894093 | 0.34398496240601506 |
|      Linear SVM     | 1.0085986491526244 | 0.33270676691729323 |
|    Random Forest    | 1.169666437206652  | 0.42105263157894735 |
|     Naive Bayes     | 1.232254762255751  | 0.39849624060150374 |
+---------------------+--------------------+---------------------+
```